Box R-GCN: Structured Query Answering Using Box Embeddings For Entities And Queries

Ruud van Bakel
VU Amsterdam
Student Number: 2614457
r.van.bakel@student.vu.nl

Michael Cochez*
VU Amsterdam
m.cochez@vu.nl

ABSTRACT

Knowledge graphs offer an efficient and straightforward way of storing information. Much of this information is typically explicitly represented in the form of a graph with nodes and edges. Graph convolutional networks (GCNs) use such an explicit graph representation to create embeddings for various tasks (e.g. link prediction, entity classification, or structured query answering). Traditional graph embeddings create such an embedding by producing a vector for each graph node. These vectors represent a point in the embedding space. Based on previous works, we introduce a model for the structured query answering task, which embeds queries and entities as boxes (i.e. axis-aligned hyperrectangles) as opposed to points. This model consists of an R-GCN and an MPQE, and uses a distance-based negative sampling loss function for boxes. Based on our test results, we could not yet conclude whether box embeddings could provide a viable way of finding multiple answers to structured queries.

I. INTRODUCTION

Information stored in knowledge graphs is explicit and accessible with relative ease. Because the information has to be stored explicitly, knowledge graphs are very susceptible to missing information. Graph embeddings can help with this problem by embedding similar entities close together, often incorporating information from neighbouring entities in the original graph. These graph embeddings do not have the explicit information from the original graphs, but instead represent the graph nodes as points in the embedding space. Apart from the entities, for the structured query answering task traditionally the queries also get embedded as points in the same embedding space.

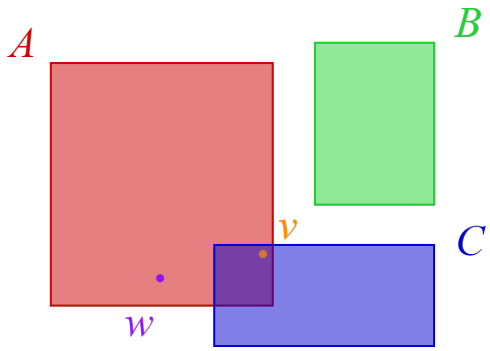

Fig. 1: A small 2D query box embedding: Here there are three queries A, B and C, and two entities v and w. In this case v is an answer to A and C, whilst w is only an answer to A.

We introduce Box R-GCN, a graph convolutional network which uses entity and query box embeddings, for the structured query answering task. Box R-GCN uses an R-GCN and MPQE model. Because of this it can treat different relation types differently (i.e. it has separate weight matrices for the different relation types).
This model uses boxes (i.e. axis-aligned hyperrectangles) to represent entities and queries in the embedding space. These boxes have multiple potential benefits over traditional "point" embeddings, as explained in the following subsections.

I-A. RESEARCH FOCUS

Our research question focuses on the potential of box embeddings for queries with multiple answers. We state our research question as follows:
Can box embeddings be used to give a finite set of answers to a query rather than a ranking of results.?
Our main focus in this work is on how models using box embeddings perform for structured query answering with multiple answers. Box R-GCN is created to provide

* Thesis supervisor

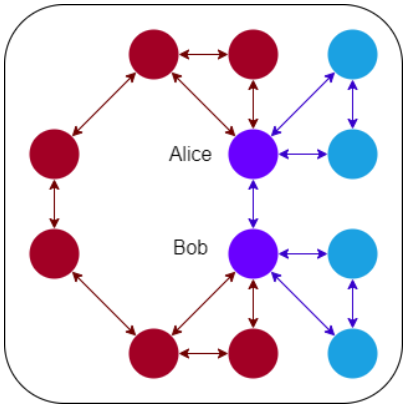

Fig. 2: Here Alice and Bob are closely related in context of the purple relations (1 relation minimum), but they are not very closely related in context of to the red relations (5 relations minimum).

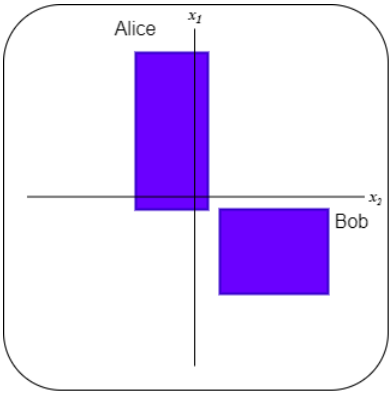

Fig. 3: Here Alice and Bob are have relatively close points (seen near the origin), but also very distant points.

such a model, which we use to evaluate the performance box embeddings.

Apart from this main focus our work also focuses on the various properties of Box R-GCN, such as the generalisability from simple query structures to more complicated structures, as seen in (Daza and Cochez 2020). Also, the properties of the generated embedding (e.g. box shapes and box locations) are interesting to analyze. The findings on these topics can be seen in section VII.

I-B. INTRODUCTION TO QUERY BOXES

Although GCNs (Kipf and Welling 2016) provide an effective way to create graph embeddings and can achieve good results on different tasks such as link prediction and node classification (Kipf and Welling 2016; Schlichtkrull et al. 2017), they do not work well for structured query answering with multiple answers.

The reason for this is that for the structured query answering task, the query is typically embedded as a point in the embedding space. The closest entity embedding to this point is then considered the answer to the query. If there are multiple answers possible however, there is no good way to determine how many of the closest points should be considered.

One potential way to solve this problem is to instead embed queries as boxes (axis-aligned hyperrectangles). Figure 1 shows such query box embeddings. These boxes can help because they provide a clear border which can be utilized to form two separate groups of entities: answers which are embedded within the box and non-answers which are embedded outside of the box. Whilst the idea of using query boxes has been experimented with before, this has to our knowledge only been done by (Ren et al. 2020). Furthermore, the idea is still very new, so further experimentation may be desirable. Finally, (Ren et al. 2020) do not actually train and test the model for multiple answers, which is our main motive for using these query boxes.

I-C. INTRODUCTION TO ENTITY BOXES

Apart from box representations for the queries, our model also uses boxes for the entities themselves. The main idea behind this is that boxes could potentially preserve information about nodes within different contexts. Figure 2 shows how two entities may be closely related in one context, but not in another. It is not possible for two entities to be embedded close to each other and far from each other at the same time if they are embedded as points. Figure 3 on the other hand shows how multiple box embeddings can have close points and distant points at the same time.

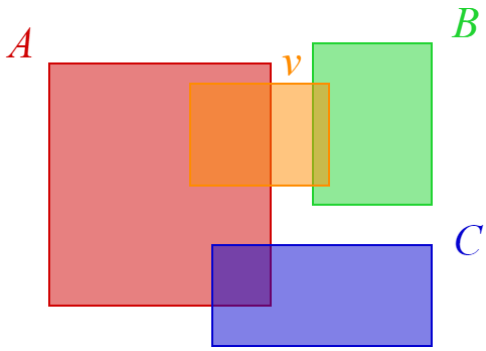

Fig. 4: A small 2D query and entity box embedding: Here there are three queries A, B and C, and one entity v. In this case v is an answer to A and B, but not to C.

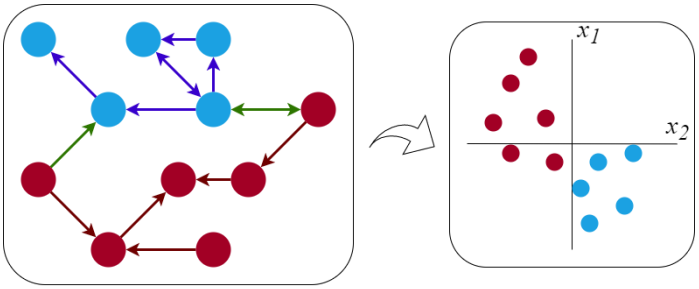

Fig. 5: Here a graph can be seen on the *left*. The different coloured arrows represent relations of different types. A GCN could turn a representation of this graph to an embedding such as represented on the *right*.

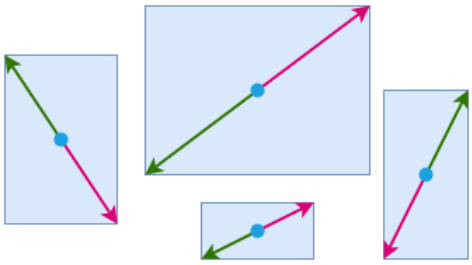

Fig. 6: Here the blue dots represent the center vectors of the boxes, the pink arrows represent the offset vectors v, and the green arrows represent opposite offset vectors $-v$. As seen here, adding and subtracting an offset vector from a center produces two opposite corners of a box.

Furthermore, as seen in figure 4 an entity with a box embedding can be an answer to query boxes that have no overlap, in contrast to point embeddings, as seen in figure 1.
This extra flexibility for the entity embeddings may help improve the graph embedding for different tasks, such as structured query answering.

II. BACKGROUND

In this section we provide some information considering the field of graph embeddings and the task of structured query answering.

II-A. MESSAGE PASSING

A message passing algorithm can include information from neighbouring nodes when creating the embedding of an entity. When calculation the next hidden representation of node, a message passing algorithm incorporates the representations of the neighbouring (i.e. directly connected to the original node in the graph) nodes. If multiple layers are used, information from nodes further away (i.e. separated by more than one edge) can influence the representation of nodes. Message passing is useful because often the relations between a node and its neighbourhood contain valuable information for its embedding.

II-B. GRAPH CONVOLUTIONAL NETWORKS

A graph convolutional network (GCN) is a graph neural network that takes a graph representation as input and produces a graph embedding as output. These embeddings typically represent nodes as points in the embedding space, by having a vector for each entity.
GCNs use a message passing algorithm for the propagation rule. Because GCNs use a message passing scheme, information from neighbouring nodes can influence the intermediate representation of a node.
The (layer-wise) propagation rule used by GCNs is:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right). \quad (1)$$

Traditional GCNs treat all edges in the graph in the same way (i.e. they share a weight matrix). However, a specific type of GCN, called the relational graph convolutional network or R-GCN (Schlichtkrull et al. 2017), can treat different types of edges differently by using different weight matrices. Figure 5 is a diagram of a graph representation turned into an embedding. An R-GCN could treat the differently coloured arrows differently, whilst a normal GCN could not. Our model uses R-GCNs for creating anchor node embeddings and query embeddings (see section IV).

II-C. STRUCTURED QUERY ANSWERING

As the name suggests, the task of structured query answering involves answering structured queries. These queries are embedded into the embedding space. Traditionally they are embedded as a point and the closest entity to this point in then considered the answer to the query. With such a setup a model is trained to embed the queries as best as possible. Traditionally such a model would use geometric interpretations of the logical operations in the structured query (Hamilton et al. 2018; Ren et al. 2020).
Our setup differs in multiple places from such a traditional setup. Firstly, as mentioned before our entities and queries are boxes. Secondly, our model does not use a pre-existing graph embedding, but instead generates one. Finally, our model uses a message passing query embedding (MPQE) model (Daza and Cochez 2020) instead of geometric interpretations. We chose the MPQE

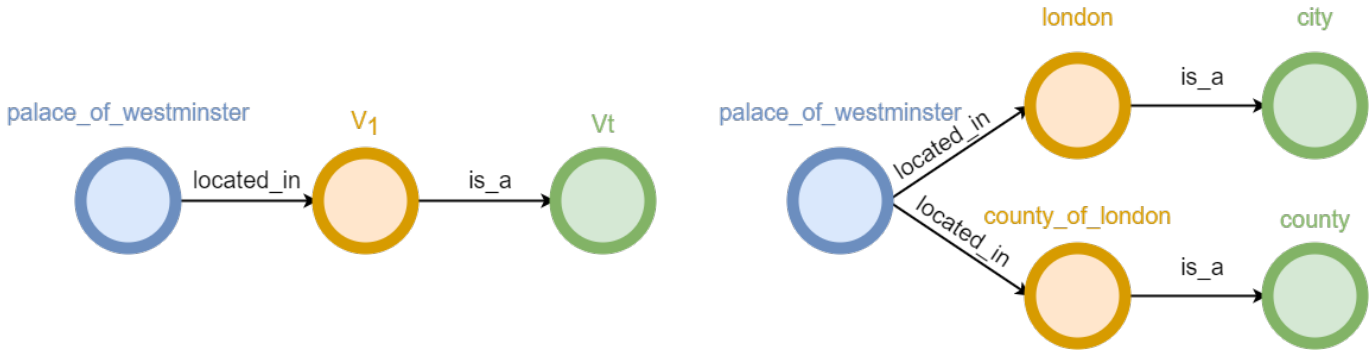

Fig. 7: *Left*: The query graph that is derived from to the query $V_t.\exists V_1, V_t : is\_a(V_1, V_t) \wedge located\_in(palace\_of\_westminster, V_1)$. The variable nodes have a specific type (e.g. *location* for V_1 and *location_type* for V_2). *Right*: A subgraph extracted from a KG. This subgraph satisfies the conditions of the query.

because it has a good reported performance (Daza and Cochez 2020) and because it can be trained on simple query structures and still be effective on more involved query structures (Daza and Cochez 2020).

III. DEFINITIONS

III-A. STRUCTURED QUERIES

For this experiment we are interested in structured queries in the *conjunctive form*. As (Daza and Cochez 2020) show, structured queries can be represented as a condition that should be met by the answers (target entities). For example, the query "select all projects *P*, such that topic *T* is related to *P*, and both *alice* and *bob* work on *T*." could be written as follows:

$$P.\exists T, P : \text{related}(P, T) \wedge \text{works\_on}(alice, T) \wedge \text{works\_on}(bob, T) \quad (2)$$

Here *alice* and *bob* are so called *anchor nodes* (i.e. actual entities in the graph) and *P* and *T* are query variables. Also seen in (Daza and Cochez 2020), the general form a structured query with a conjunctive form is as follows:

$$q = V_t.\exists V_1, \ldots, V_m : r_1(a_1, b_1) \wedge \ldots \wedge r_m(a_m, b_m), \quad (3)$$

where $r_i \in \mathcal{R}$, and a_i and b_i are either entities in the set of all graph entities $\mathcal{V}$, or query variables in $\{V_t, V_1, \ldots, V_m\}$. We will use this as the definition for our structured queries.

III-B. BOXES

As mentioned before, traditionally entities and queries get embedded as a point. Such a point is represented with a single vector. There are multiple ways to describe a box (i.e. axis-aligned hyperrectangle) in an embedding space. To describe all possible boxes in an embedding space with n dimensions, at least two vectors $v1, v2 \in \mathcal{R}^n$ are required. One way to use these vectors to represent a box is by letting one represent the *center* of the box and by letting another be an *offset* vector (Ren et al. 2020). As shown in figure 6, when this offset vector is added to the center vector, one corner of the box is produced and when subtracted from the center the opposite corner is produced. So the center vector effectively determines the location of the box, whilst the offset vector effectively determines the size and shape of the box.

III-C. ANSWER FUNCTION

Since we also embed entities as boxes, our model can not use the answer condition used by (Ren et al. 2020), where answers are points embedded within the query box, and non-answers are embedded outside of the query box. Instead our answer condition is based on what entities overlap with a query box. If an entity box has any overlap with a query box then it is considered an answer to the query.
Allowing any amount of overlap (even with a volume of 0) to be sufficient gives relatively much flexibility to the embedding. In contrast if total overlap were to be required (i.e. an entity box has to be fully embedded within a query box to be considered an answer), likely a more stable, but less flexible embedding would be created.

IV. THE MODEL

Our model consists of three main parts: stored entitiy embeddings, an R-GCN, and an MPQE model. The model is designed to be trained and tested per query.

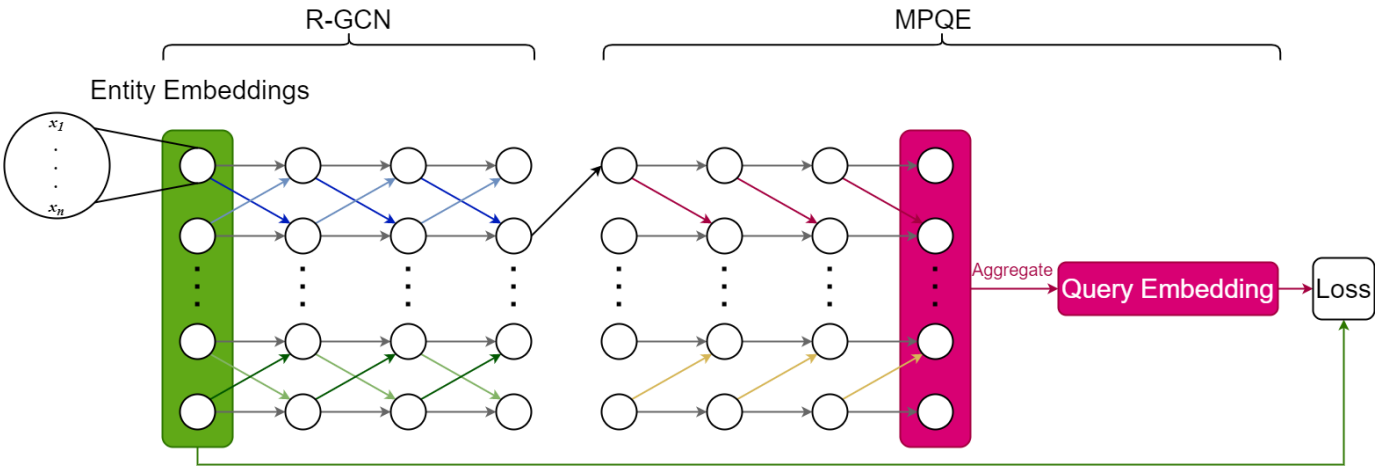

Fig. 8: Diagram of the full model. First the entity embeddings are given as an input to the R-GCN. The outputs of the R-GCN representing the query anchor nodes are then passed to the MPQE model. This model then uses these anchor embeddings along with generic type embeddings to produce a query embedding. The entity and query embeddings are then used to calculate the loss.

Furthermore, a special loss function is used for training the model.

IV-A. ENTITY EMBEDDINGS

The R-GCN model takes node embeddings as an input. These node embeddings are remembered and are trained with the model through backpropagation.

As mentioned before, each entity embedding consists of a center vector and an offset vector. This setup allows us to sample the initial center and offset vectors from different distributions. We have chosen to sample the center vectors from a uniform distribution. This allows for the initial entity embeddings to be scattered evenly over the embedding space. The offset vectors on the other hand are sampled from a normal distribution. This makes it so that the boxes initially tend to have a similar size. Of course, the shapes and locations could drastically vary after having been trained for some time.

IV-B. THE R-GCN

Our model uses an R-GCN to generate anchor node embeddings from entity embeddings. This R-GCN has separate weights for each relation type. As (Schlichtkrull et al. 2017) show, the message-passing update rule for the R-GCN is:

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right), \quad (4)$$

where $\mathcal{N}_i^r$ denotes the set of neighbour indices of node i under relation $r \in \mathcal{R}$. $c_{i,r}$ is a problem-specific normalization constant that can either be learned or chosen in advance (such as $c_{i,r} = |\mathcal{N}_i^r|$). Here the $\sum_{r\in\mathcal{R}}$ and $\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)}$ parts are responsible for the separate weight matrices for the different relations.

The way the layers are connected is dependent on the current query. A sample of negative targets (i.e. entities which are not an answer to the query) is taken from around the anchor nodes of the query. The edges connecting these nodes to the anchor nodes are then used to connect the R-GCN. The inverse of these edges are also used for the R-GCN. For example, if the edge *(alice, relation_1, bob)* is sampled, then the connections corresponding to *(alice, relation_1, bob)* and *(bob, relation_1_inv, alice)* are present within the R-GCN. The reason these edges are chosen instead of the edges in the subgraph corresponding to the query graph (see figure 7) is because some queries have an enormous corresponding subgraph. Such a subgraph could contain too many edges for the model to train in a timely manner.

For our model the size of the hidden representations does not change throughout the layers of the R-GCN.

For the non-linearity σ between the layers a ReLu function is used, as it does not suffer from vanishing gradients as the sigmoid function does.

Finally, at the last layer of the R-GCN, anchor node embeddings are produced. These are actually simply entity embeddings, but only the ones representing anchor nodes in the current query are passed on to the MPQE model.

IV-C. THE MPQE MODEL

Once the anchor nodes are generated the MPQE model can start with generating the query embedding. Apart from these anchor embedding, this model also uses generic type embedding for the query variables (Daza and Cochez 2020). These generic type embeddings are generic embeddings that represent variable nodes with a specific type. Similarly to the entity embeddings, these embeddings get stored and are trained with backpropagation.

This MPQE model then uses all the anchor and type embeddings as an input for an R-GCN. The embeddings and weights are shared for the different query structures (see figure 10).

The output of the final layer of this R-GCN gets aggregated to form the final query embedding. This query embedding is then used in the loss function along with the entity embeddings.

IV-D. LOSS

Our model uses a distance-based negative (Sun et al. 2019) sampling (Mikolov et al. 2013) loss function. A distance metric is required to define what distance will be calculated for the loss function. For our distance metric we will use the Manhattan distance between the border of an entity box and the center of a query box (see figure 9). Such a Manhattan distance metric has already been proven effective for graph embeddings with queries as boxes and entities as points (Ren et al. 2020).

Heavily inspired by (Ren et al. 2020), our distance metric is defined as follows:

$$\begin{aligned} \text{dist}_{\text{box}}(\mathbf{e};\mathbf{q}) = \text{dist}_{\text{outside}}(\mathbf{e}\ ;\ \text{closest}(\mathbf{e};\mathbf{q})) + \\ \alpha \cdot \text{dist}_{\text{inside}}(\mathbf{e}\ ;\ \text{closest}(\mathbf{e};\mathbf{q})), \end{aligned} \tag{5}$$

where $0 < \alpha < 1$ is a fixed scalar factor and $closest : b_1, b_2 \mapsto v$ is a function that takes two boxes, b_1, b_2, and returns the closest point v from b_1 to the center of b_2. If b_1 and b_2 do not overlap, this point is also one of the points of the surface of b_1 that is closest to the surface of b_2.

It is important that α is used to downweight the inside distance. This is important because once an entity box has some overlap with a query box (i.e. the outside distance is 0), it is already considered to be an answer. Being even closer to the query box center does not make an entity more of an answer.

Furthermore, $\text{dist}_{\text{outside}}$ and $\text{dist}_{\text{inside}}$ are defined the same as in (Ren et al. 2020):

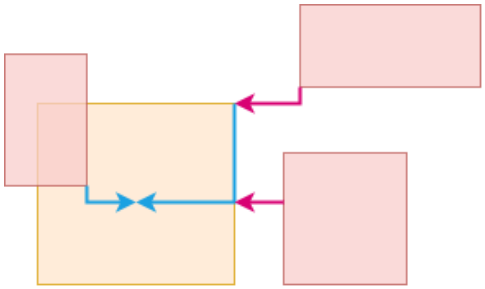

Fig. 9: Here the orange box is a query box and the red boxes are enitity boxes. The pink arrows represent the outside distance, whilst the blue arrows represent the inside distance.

$$\begin{aligned} \text{dist}_{\text{outside}}(\mathbf{v};\mathbf{q}) = \\ \|\text{Max}(\mathbf{v} - \mathbf{q}_{\text{max}}, \mathbf{0}) \ + \ \text{Max}(\mathbf{q}_{\text{min}} - \mathbf{v}, \mathbf{0})\|_1 \end{aligned} \tag{6}$$

$$\begin{aligned} \text{dist}_{\text{inside}}(\mathbf{v};\mathbf{q}) = \\ \|\text{Cen}(\mathbf{q}) - \text{Min}(\mathbf{q}_{\text{max}},\ \text{Max}(\mathbf{q}_{\text{min}}, \mathbf{v}))\|_1 \end{aligned} \tag{7}$$

where $Max : v_1, v_2 \mapsto v_{out}$ and $Min : v_1, v_2 \mapsto v_{out}$ represent functions that map two vectors v_1, v_2 to one vector v_{out} in which each element v^i_{out} is equal to respectively the maximum or minimum of v^i_1 and v^i_2. Furthermore, $\| \ \|_1$ represents the L1 norm of a vector.

The loss function itself is a negative sampling loss function (Mikolov et al. 2013). Given the distance metric, our loss is defined as follows:

$$\begin{aligned} L = \\ -\sum_{i=1}^{k} \frac{1}{k} \log \sigma(\log(\gamma) - \log(\text{dist}_{\text{box}}((\mathbf{e};\mathbf{q})) + \omega)) \ - \\ \sum_{i=1}^{k'} \frac{1}{k'} \log \sigma(\log(\text{dist}_{\text{box}}(\mathbf{e}'_{\mathbf{i}};\mathbf{q}) + \omega) - \log(\gamma)), \end{aligned} \tag{8}$$

where $e \in [\![q]\!]$ is an answer to the query and $e'_i \notin [\![q]\!]$ is not an answer to the query. Also γ is a fixed scalar margin, k is the number of positive entities, k' is the number of negative entities, and ω is a small offset to prevent the logarithm around the distance from being undefined when an entity box overlaps with a query center. Here we chose σ to be the sigmoid function to stay more in line with the loss function from (Ren et al. 2020). Since this is the only part in our model where we use the sigmoid function this is unlikely to result in vanishing gradients.

We do not backpropagate the gradients directly from the loss, but we instead use the Adam optimizer (Kingma and Ba 2014), as it has as been proven to be an effective optimizer in the past (Kingma and Ba 2014; Jais et al. 2019).

IV-E. UNSEEN NODES

It is possible that during training our model never sees certain nodes. This would lead to these nodes still using their initial embedding after training. This initial embedding may not contain any useful information for its entity, but the neighbourhood of this node still might. Despite not being trained, these initial embeddings could still be turned into something useful as the message passing algorithm in the R-GCN model incorporates information from the neighbourhood.

V. RELATED WORK

Currently not much research has been performed on the topic of box embeddings. Our approach to boxes is mostly inspired by (Ren et al. 2020). In (Ren et al. 2020) they embed the queries as boxes, but not the entities. Although they use a similar loss function and distance metric, their approach to creating the query embeddings is vastly different from ours. They use a more traditional approach of using geometric interpretations of the query operations (Hamilton et al. 2018), where we use an MPQE model. Also they do not train and test their model on queries with multiple answers.

Although less relevant for our work, in (Vilnis et al. 2018) a class of models that assign probability measures to order embeddings is described. In their model they embed entities as *box latices*. Although we have not taken much inspiration from their work, it is still one of the rare papers that uses box embeddings and the only paper we could find which used box embeddings for graph entities.

In (Daza and Cochez 2020) the message passing query embedding (MPQE) model is introduced. As mentioned before, we use this model for generating our query embedding using anchor node embeddings provided by an R-GCN and generic type embeddings.

Staying on the topic of models we used, in (Schlichtkrull et al. 2017) the relational graph convolutional network (R-GCN) is introduced. We use this model for generating anchor node embeddings. Furthermore, an R-GCN is used in the MPQE model.

These models we described (R-GCN and MPQE) both use a message passing framework. In (Gilmer et al. 2017) multiple message passing neural networks are described. The message passing framework used in (Schlichtkrull et al. 2017), is inspired by special cases of a simple differentiable message-passing framework (Gilmer et al. 2017).

The actual concept of GCNs was introduced in (Kipf and Welling 2016). As the name suggests, the R-GCN is based on GCNs. This in turn makes our model also based on GCNs.

Although we use an MPQE model for generating our queries, the method of training geometric interpretations of the query operations to create graph embeddings (Hamilton et al. 2018) has been around longer and has therefore also currently been used more often in research than the MPQE model.

	AIFB	MUTAG	AM	Bio
Entities	2,601	22,372	372,584	162,622
Entity types	6	4	5	5
Relations	39,436	81,332	1,193,402	8,045,726
Relation types	49	8	19	56

TABLE I: Statistics of commonly used knowledge graphs for graph representation learning and query answering. We only use the AIFB dataset. From (Daza and Cochez 2020).

VI. EXPERIMENTS

In our experiments we only consider the seven query structures seen in figure 10. These specific query structures have been used before often (Ren et al. 2020; Daza and Cochez 2020; Hamilton et al. 2018). Another reason for just focusing on these query structures is that these structures capture most commonly used queries (Arias et al. 2011).

VI-A. DATA SETS

For these experiments we will use the AIFB data set. This data set is publicly available and has been used before for graph representation learning (Ristoski et al. 2016; Ristoski and Paulheim 2016; Schlichtkrull et al. 2017) and query answering (Daza and Cochez 2020). The statistics of this data set can be seen in table I.

- **AIFB:** A knowledge graph of academic institution. The entity types in this knowledge graph are persons, organizations, projects, publications, and topics.

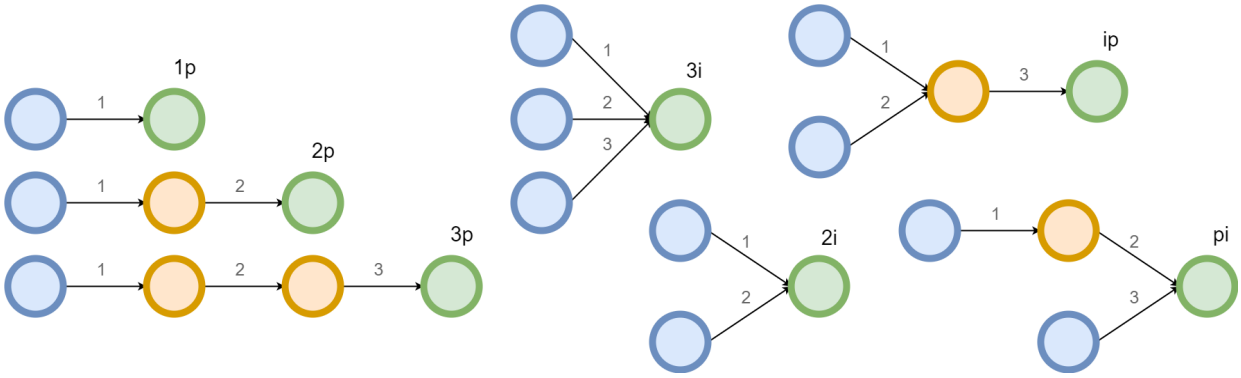

Fig. 10: The seven query structures we use. The blue circles are anchor nodes, the orange circles are variable nodes and the green nodes as (variable) target nodes. The same names are used here as were used in (Ren et al. 2020).

VI-B. SAMPLING CONTROL

As mentioned in subsection IV-D, our model uses a negative sampling loss. Our loss function can handle multiple positive answers (i.e. actual query answers) and multiple negative answers (i.e. entities that are not answers to the query) for each query. Because we are interested in queries with multiple answers, the possibility to use multiple positive answers is an interesting option to explore.

Our model only trains the entities sampled for the loss function. Because of this, we can control the ratio of trained positive answers to trained negative answers. In the next subsection we describe how the amount of positive and negative samples is determined for each sampled query.

VI-C. NEW CHALLENGES

Since our model uses an MPQE model for embedding the queries, sampling in the same way as in (Daza and Cochez 2020) could seem like a good idea. There are a few complication for our model when applying such an approach.

Firstly, since we are interested in creating a model that can handle queries with multiple answers, we will need to find all the answers to a query for at least the validation set and the test set. Without knowing all the answers there is no good way to judge the performance of the model for validation and testing. Although it is in principle not required, we also opted to use multiple positive answers during training.

Secondly, our model also uses an R-GCN to create anchor node embeddings which are passed to the MPQE model. The way in which the layers in this part of the model should be connected should also be sampled with the query. Theoretically the R-GCN could be connected based on the full model. This way would not required a sample per query, as the connections in the graph stay constant. For computational efficiency we opted to instead sample a subgraph from the full graph for each query.

Apart from these additions our query sampling method is very similar in effect to the one used in (Daza and Cochez 2020).

VI-D. TRIVIAL SATISFACTION

The initial graphs we use are directed. Before actually searching for queries we create a graph that is that same as the initial graph, but with the inverse edges added. Our target node search algorithms can use the same edge type and variable node type in different places. For example, a query with the edges $((alice, has\_friend, v_1), (v_1, has\_friend, v_2))$, where v_1 and v_2 have the same type.

Because our graph has the inverse edges added, any query structure could be found by simply going back and forth between two connected nodes. Figure 11 give an example of such a trivially satisfied query. Because a simple searching algorithm could suffer from this trivial satisfaction our algorithms have specifically take this into account.

Figure 12 show how sometimes seemingly trivial connections may not actually be trivial.

VI-E. SAMPLING ALGORITHM

Our sampling algorithm we use to find queries with the structures seen in figure 10 starts by sampling an edge in the graph. From this edge it searches for one of the aforementioned query structures. It does this by looking at all candidates for the next edge and then selecting one. For each chosen edge, its type and the type of the node it is going towards are used to build the query. Whilst

Clock	Scratch	Memory	Sockets	Cache	Cores	GPUs	Interconnect
1.70 GHz	1.5 TB NVME	256 GB UPI 10.4 GT/s	2	8.25 MB	12	4 x GeForce 1080Ti,11GB GDDR5X	40 Gbit/s ethernet

TABLE II: The specifications of the hardware available to us on the Lisa system. The models are trained, validated and tested on the scratch file system.

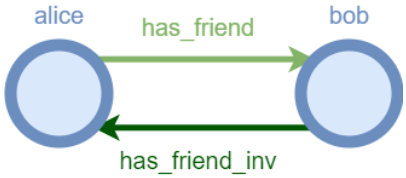

Fig. 11: If a 3p search algorithm selects the has_friend edge it could find the following edges: $((alice, has\_friend, bob), (bob, has\_friend\_inv, alice), (alice, has\_friend, bob))$. There is no real 3p structure here, but the inverse edges create these trivial cases.

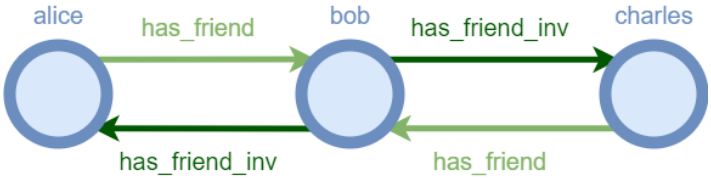

Fig. 12: If a 2p search algorithm selects the has_friend edge between *alice* and *bob* it could generate the following query: $P.\exists V_1, V_2, P : \text{has\_friend}(alice, V_1) \wedge \text{has\_friend\_inv}(V_1, V_2)$. Although this query can be trivially satisfied, it can also be satisfied in a non trivial manner since bob as an outgoing has_friend_inv edge that does not go back to *alice*.

doing this, the algorithm also finds all the answers to the query. The algorithm also makes sure it does not sample duplicate queries. The full algorithm is described in more detail in the appendix.

VI-F. TRAIN, VALIDATION AND TEST SETS

Before actually sampling queries some edges are removed form the original graph. Of the original edges 90% remain in the graph and 10% are put away for the validation and test set.
This now smaller graph is used to sample the queries for the train set from. This sampling happens as described before.
For the validation and test set first an edge is sampled from the 10% removed edges. The algorithm for sampling validation and test queries is mostly the same as the algorithm for the train queries, with the exception that this sampled edge must be included at some point within the sampled subgraph. This makes sure that every query in the validation and test set contains information not seen during training.
A set of queries is sampled in this manner. Of these queries one eleventh is used for the validation set, whilst the remaining queries are used for the test set. Since all these queries are unique, no query appears in both the validation and test set.
When validating and testing the first R-GCN is connected based on the edge of the whole graph, as opposed to a sample.

VI-G. IMPLEMENTATION

Our model and the query generation code is implemented in python. We use the R-GCN implementation from pytorch geometric. The MPQE code we used is found on github (Daza and Cochez 2020). The rest of the code is implemented by ourselves and is also available on github[1]. The code is written for in python 3.7.7 for pytorch 1.5.0, with cuda 10.1.
These jobs were performed on the Lisa system[2] from SURFsara. The specifications of the hardware we used on the Lisa system can be seen in table II.

VI-H. EXPERIMENTS

We perform two experiments on the AIFB data set. In the first experiment we sample 8750 queries with structures seen in figure 10. This number is chosen as 8750 queries roughly contain a million targets. This model is trained for one epoch.
For the second experiment we sample 1250 queries, but this time the queries only have the 1p structure. This model is trained for seven epochs.
Both the validation and test set contain queries of all seven structures seen in figure 10. For the validation and test set 100 queries are sampled, 9 of which go to the validation set. The remaining 91 queries go to the test set. The same test set is used for both experiments.

[1] github.com/R-van-Bakel/BPAI_Box-Embeddings
[2] https://userinfo.surfsara.nl/systems/lisa

		predicted value p	predicted value n	total
actual value	p′	13,410	449	13,859
	n′	206,820	18,613	225,433
	total	220,230	19,062	

Fig. 13: Results of experiment 1. This model has a precision of 0.061, a recall of 0.968 and an F1 score of 0.115.

		predicted value p	predicted value n	total
actual value	p′	2,704	11,155	13,859
	n′	131,221	94,212	225,433
	total	133,925	105,367	

Fig. 14: Results of experiment 2. This model has a precision of 0.020, a recall of 0.195 and an F1 score of 0.037.

Hyperparameter tuning in only performed for the first experiment. The chosen hyperparameter settings will then be used for both experiments. This also means that we only use the validation set for the first experiment. The final hyperparameter settings can be found in the appendix.

VII. RESULTS

Since using query boxes to find multiple answers to queries has not been done before to our knowledge, we do not have a proper baseline to compare our results to. This may prove not to be a large problem as our focus is not on getting state of the art results, but on figuring out if query boxes could offer a viable way of giving a finite set of answers to a query.

Figure 13 shows the results of the first experiment. With a total predicted amount of positive targets of 220,230 and a a total predicted amount of negative targets of 19,062 the model seems very eager to consider an entity an answer to a query. When considering that there are much fewer actual positive targets than actual negative targets, the model does not seem to perform very well. Due to the model to predicting positive targets far to often, the precision of the model is very low, but the recall if very high. Because of this low recall, the F1 score for the model is also quite low.

As the model performs poorly, which is trained on 8750 unique queries of all seven structures, it is no surprise that the model which is trained on 1250 unique queries with only one query structure performs very poorly. As figure 14 shows, the second experiment did not yield good results.

Table III shows some statistics about the trained entity boxes. This table shows the averages of the standard deviation and mean of all dimensions of the center and offset vectors. These values did not tend to vary much among the dimensions of the respective vectors. For example, the standard deviations for the dimensions of the center vectors from the first experiment were all between 55.9 and 58.1. As can be seen in table III, the query boxes tended to have a similar size, but were still very spread around.

VIII. DISCUSSION

There are multiple possible explanations for why the results of the first experiment turned out the way they did. The actual reason is likely a combination of these explanations.

Firstly, the model considered entities an answer to a query if their embeddings had any overlap with the query

	Avg. Std.	Avg. Mean
Experiment 1 Center	56.962	-0.145
Experiment 1 Offset	1.960	4.159
Experiment 2 Center	56.941	0.041
Experiment 2 Offset	1.963	4.155

TABLE III: The average standard deviations and means of the dimensions of the center and offset vectors for experiment 1 and experiment 2.

embedding. This relatively liberal criterion most likely makes it very easy for entities to be considered an answer to a query. A more conservative criterion requiring more overlap could make the model predict negative targets more often.

Secondly, we performed a relatively little hyperparameter tuning. This may however likely not be a major cause for the final results as the model performed very similarly under the different hyperparameter settings.

Thirdly, the model is trained on considerably fewer queries that what is typically the case. The reason for this is that our model trains on multiple targets per query, but it could be possible that this has a negative effect on the model convergence. Such conclusions are still difficult to make, as we have not found any research on this topic.

Fourthly, our loss function may simply not effect the negative targets as much as it should. An alternative loss function that would put more emphasis on moving the wrong entity boxes away from a query box, could potentially help with this problem.

Finally, this topic has not been studied much as of yet. Because of this it is difficult to make many predictions about the model in advance. This topic still requires more research to be properly understood. Our setup was very experimental, which should not make relatively bad results a surprise. To gain better insight into the potential of query and entity boxes more research is in order.

Apart from these explanations there is more to discuss regarding the results. One thing to note is that table III suggests that the entity boxes were not very big and were spread out. This could suggest that the query boxes for experiment 1 must have been quite large in order to predict that amount of positive targets.

As for the second experiment the worse conditions (i.e. less unique queries and only one used query structure) are probably the reason the model performs worse that the model from the first experiment. Due to the poor results there is not much more interesting to say about the performance in experiment 2.

IX. CONCLUSION

We have introduced Box R-GCN, a model that uses axis-aligned hyperrectangles to represent knowledge graph entities and structured queries. The results of our experiments were quite poor and warrant further experimentation.

Although the results are not conclusive there are still many options available for different variants of the model which may perform better. For example, the answer function and the loss function could be changed to something entirely different. Another part to experiment with is the amount of answers used to train each query. Right now our model uses all answers to a query for training. Models that are not made to find multiple answers to a query often train with only one target per query. Our model could also be modified to do that. It could than train on more queries, but with only one target used for training per query. Our model could also be modified to not use the full set of answers for training, but to instead use a sample of these answers.

For now there are many ways to still experiment with this model. If after some more experimentation better results are achieved, it would be a good idea to also test it on other data sets such as the ones seen in table I.

Altogether, with the result from both experiments, we can not yet conclude whether box embeddings can be used to give a finite set of answers to a query rather than a ranking of results.

REFERENCES

Arias, M., Fernández, J. D., Martínez-Prieto, M. A., and de la Fuente, P. (2011). An empirical study of real-world sparql queries.

Daza, D. and Cochez, M. (2020). Message passing query embedding. In *ICML Workshop - Graph Representation Learning and Beyond*.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry.

Hamilton, W. L., Bajaj, P., Zitnik, M., Jurafsky, D., and Leskovec, J. (2018). Embedding logical queries on knowledge graphs.

Jais, I. K. M., Ismail, A. R., and Nisa, S. Q. (2019). Adam optimization algorithm for wide and deep neural network. *Knowl. Eng. Data Sci.*, 2(1):41–46.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

Ren, H., Hu, W., and Leskovec, J. (2020). Query2box: Reasoning over knowledge graphs in vector space using box embeddings.

Ristoski, P., de Vries, G. K. D., and Paulheim, H. (2016). A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., and Gil,

Y., editors, *The Semantic Web – ISWC 2016*, pages 186–194, Cham. Springer International Publishing.

Ristoski, P. and Paulheim, H. (2016). Rdf2vec: Rdf graph embeddings for data mining. In Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., and Gil, Y., editors, *The Semantic Web – ISWC 2016*, pages 498–514, Cham. Springer International Publishing.

Schlichtkrull, M., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I., and Welling, M. (2017). Modeling relational data with graph convolutional networks.

Sun, Z., Deng, Z.-H., Nie, J.-Y., and Tang, J. (2019). Rotate: Knowledge graph embedding by relational rotation in complex space.

Vilnis, L., Li, X., Murty, S., and McCallum, A. (2018). Probabilistic embedding of knowledge graphs with box lattice measures.

Appendix

SAMPLING ALGORITHM

Our sampling algorithm is implemented separately for each considered query structure (see figure 10). All these implementations start at some random edge in the graph. For each implementation the edge labeled as *1* in figure 10 is sampled first. Since the graph is directed the first anchor node can be determined based on the direction of the edge. Once such a beginning has been found the algorithm for finding all target nodes can continue expanding from there. During this search the actual query is also generated. One important property shared between all the algorithms is that the initial edges are sampled directly out of a set of edges. Another approach could be to first sample an anchor node and to afterwards sample one of its outgoing edges. Our approach make any edge equally likely to be initially sampled, but this also means that nodes with many edges connected to it are more likely to be sampled as an anchor node or for the first query variable.

For the **1p** query structures the algorithm is very simple. Once the initial edge is sampled, the first target node can also be found as it is the node the edge is going towards. After that, the other edges from the anchor node leading to answers are searched. These edges need to have to have the same type as the initial edge and the type of the potential target needs to have the same as the initially found target node. During this search the target nodes are added to the set of target nodes $target$.

The **2p** query structures the algorithm is very similar to the 1p algorithm. First the whole 1p algorithm is performed, but instead of a target set it produces a set of variable candidates. Now an edge search is performed. A set $edge_2$ is created of all outgoing edges from these nodes. One of these edges is then sampled to provide the type of the second edge and the type of the target. If this edge happens to have the inverse type of the previous edge $t1\_inv$, then a new search is performed. This search traverses all edges in $edge_2$. If any edge is found of type $t1\_inv$ which does not go to the anchor node, the edge type and target type are accepted. This is done to work around the trivial satisfaction problem. Figure 12 gives an example of such a case. Once a type for the second edge and a type for the target node have been found, the other edges in $edge\_2$ with the same type and target type are searched. The target nodes are then added to a set $target$.

Similar the to the p2 algorithm, the **3p** algorithm is simply an extension of the previous one. The p3 algorithm is in fact the same as the p2 algorithm, except that another edge search is performed before creating the target set. This extra edge search only differs from the previous one in how it deals with the trivial satisfaction problem. In the p2 algorithm the anchor node was used to determine whether an edge lead to trivial satisfaction. As can be seen in figure 10, the third node is not preceded by an anchor node. Since we combine all variable nodes in sets, we have no information about the exact path taken to reach later variable nodes. We only know that a variable node is connected to some node in the previous variable set or to the previous anchor node. Because of this, to prevent trivial satisfaction the p3 algorithm does not look if an edge goes to the previous anchor node, but it looks if it goes to some node in the previous variable set. If there is one edge for which this is not the case the relation type and target type are accepted.

The **2i** algorithm starts by first performing the the p1 algorithm. Then a set of outgoing edges is generated form the variable nodes. One of these edges is sampled to find the second anchor node, if this edge would lead to trivial satisfaction then another edge is sampled until there is no trivial satisfaction. The inverse of this edge will be the second edge for the query. Now all outgoing edges of this type from the second anchor node are searched. This produces a set of nodes. To get the final target set, the intersection between this set and the first variable set is taken.

The **3i** algorithm is simply an extended 2i algorithm. Once the set intersection S is generated a third edge is sampled from this in a similar manner as the second edge, except now trivial satisfaction can occur from a relation back to the first or second anchor node. Once an eligible edge is found the third anchor node is found. The inverse of this edge is then selected as the third edge for the query. Now just as with the previous anchor nodes a set is created with this anchor node and relation. The intersection of this set and S then contains the final target nodes.

The **ip** algorithm starts out with the 2i algorithm. Then it performs an edge search like in the 2p algorithm. The only additional thing in this algorithm is that trivial satisfaction has to be prevented for with regards to both anchor nodes.

Finally, the **pi** algorithm starts out with the 2p algorithm. From this point on the algorithm continues to search for a third edge and second anchor node, just as in the 2i algorithm does for its second edge. To prevent trivial satisfaction the same rules as in the 3p algorithm are used.

One final thing to notice is that if at any point in any of the algorithms no edge can be found to further the algorithm, the whole algorithm starts over again.

HYPERPARAMETER SETTINGS

• **Embedding Dimension**	128
• **Number of RGCN layers**	3
• **Number of MPQE layers**	3
• **Learning rate**	0.0001
• **Weight Decay**	0.0005
• **Number of bases RGCN**	98
• **Number of bases MPQE**	98
• **MPQE readout**	mp
• **MPQE scatter operation**	add
• α	0.2
• γ	24
• ω	0.001