Human-in-the-loop Entity Set Expansion using Knowledge Graphs

Péter Kardos, András London and Richárd Farkas

University of Szeged, Szeged, Hungary

Abstract

Knowledge Graphs have the ability to represent our world in intricate detail within a structure that can be interpreted by both humans and machines. They not only provide descriptions of individual entities, but also indicate their interrelationships, thus serving as an effective resource for discovering semantic similarities. In this paper, we address the task of Entity Set Expansion, where the goal is to retrieve additional entities from a Knowledge Graph, given a semantically connected input seed set of entities. We developed a graph walk-based algorithm capable of recognizing and explaining the connections found between the seed elements, then it carries out the set expansion by these connections. As a set of entities usually have multiple plausible semantic connections, the user wants to choose among them. We simulated a human-in-the-loop system that allows users to influence the output of the system. In addition, we explored two baseline approaches: a purely text-based approach, using sentence transformers along with popular Large Language Models. We also proposed human-in-the-loop approaches for both baselines. We evaluated and compared all three solutions on a task derived from the KGQA LC-QuAD dataset, pointing out a huge difference in performance in favor of our graph walk-based solution. Additional experiments showed intriguing results regarding human involvement in the selection of connections within the graph walk approach and highlight the problem of the overly generalized perspective of Large Language Models.

Keywords

Entity Set Expansion, XAI, Knowledge Graph, Large Language Models

1. Introduction

Let us consider the problem where the user has a small entity set `{BMW, Volkswagen, Audi}` and wants to list more elements that are semantically related. This task is called Entity Set Expansion (ESE) [1]. The extended set can be beneficial for various downstream tasks such as Taxonomy Construction [2] or Semantic Search [3].

Considering the ESE example of `{BMW, Volkswagen, Audi}`, it is anticipated that there are connections to `car manufacturers`, and to `Germany` as well. At this point, we would need more information if the extended list is either `German car brands`, `Car brands` or `German brands` which can only be provided by the user. In order to ease the challenge of this ambiguity, we propose three human-in-the-loop assisted approaches and simulate the ability to interact with the users for the models. Each of our solutions can provide options for the user to choose from that can help disambiguation to achive an extended set. Human feedback can be leveraged in various ways. For example, reinforcement learning from human feedback (RLHF) has recently gained fame in the training of large language models (LLM), as demonstrated by systems such as ChatGPT [4].

Knowledge Graphs (KGs) have the ability to represent and organize entities and their connections in a structured way that is readable by both humans and machines, making them appropriate building blocks for human-in-the-loop ESE methods. It has been shown that KGs can boost the performance of NLP applications in tasks such as Reasoning [5] or Text Generation [6]. KGs such as DBPedia [7], YAGO or FreeBase reach substantial size as they contain fine details about their entities that can help AI systems to better understand the semantic relations between concepts. ESE from KG (KG-ESE) is

XAI-KG'25: 1st International Workshop on Explainable AI and Knowledge Graphs (XAI+KG), June 01-05, 2025, Portoroz, Slovenia
kardos@inf.u-szeged.hu (P. Kardos); london@inf.u-szeged.hu (A. London); rfarkas@inf.u-szeged.hu (R. Farkas)
0000-0003-4896-6755 (P. Kardos); 0000-0003-1957-5368 (A. London); 0000-0001-7019-2632 (R. Farkas)

the task of expanding a given set of entities by identifying and incorporating additional, contextually relevant entities contained within a Knowledge Graph.

The main contribution of this paper is to propose a KG-based ESE algorithm that outputs possible human-interpretable semantic connections among entities besides the expanded entity set, thus providing a natural way of interaction with the human user.

Since the available ESE benchmark datasets are all corpus-based, we had to create a KG-linked ESE task to eliminate the KG entity-linking errors. We used a KGQA dataset, namely LC-QuaD [8] and ran its SPARQL queries over DBPedia to acquire the extended sets and created the seed sets by sampling.

We propose two other approaches for the KG-ESE task. First, we leverage sentence transformers to generate dense vector embeddings for each entity within the KG and use vector similarity search to expand the initial set of entities. Furthermore, we introduce both heuristic-based and human-involved approaches for dimension weighting, aiming to enhance the refinement of the similarity metric. In our second approach, we utilize the widely adopted GPT-4o generative AI model to address the KG-ESE task, while also conducting experiments to evaluate the model's capacity to explain the semantic relationships between the provided entities. We comparatively evaluate the three approaches on the evaluation dataset derived from LC-QuaD. In all of the three cases, we also simulate the human-in-the-loop use case and present empirical results on these simulations.

2. Methods

2.1. Problem definition

We define the Entity Set Expansion (ESE) problem as the continuation of an initial seed of entities, by listing additional entities. For instance, given the seed `{Budapest, Bucharest, Belgrade}` one can continue with `Bratislava, Zagreb, Prague` if the presumed association is `Eastern European capitals`. However, multiple correct continuations can be correct such as capitals starting with the letter B, leading to ambiguous results.

Formally, given a seed set $S = \{s_1, s_2, ..., s_k\}$ where s_i is an entity from $KG = \{(h, r, t) \in V \times R \times V\}$ where V is the set of entities, R is the set of relations, the goal is to expand the list with additional elements arriving at the target set $T = \{t_1, t_2, ..., t_n\}$ where $S \subseteq T, t_i \in V$.

We introduce a graph walk-based algorithm and evaluate its performance against traditional text embedding-based methods as well as against a widely used large language model, GPT-4, through prompting.

2.2. Truncated Graph Walk

Our hypothesis is that the initial seed elements have a common ancestor that we can reach by traversing the same type of edges.

We propose an algorithm that not only extends the list of seed nodes, but also provides an explanation of how they are connected by listing paths that lead to common ancestors.

This solution was developed utilizing graph walks as the central methodology. We predefined path templates that the algorithm uses to search for common connectivity paths in the graph. We restricted each path to take a route over the same type of edges and to arrive at the same node. We call this node the common ancestor or endpoint ep for short. An ancestor does not necessarily have to be higher in the hierarchy as there may be no hierarchy in the KG.

The predefined templates of length one and length two are the following:

1. Forward (F): $(s \rightarrow r_1 \rightarrow ep)$
2. Backward (B): $(ep \rightarrow r_1 \rightarrow s)$
3. Forward-Backward (F-B): $(s \rightarrow r_1 \rightarrow Y) \wedge (ep \rightarrow r_2 \rightarrow Y)$
4. Forward-Forward (F-F): $(s \rightarrow r_1 \rightarrow Y) \wedge (Y \rightarrow r_2 \rightarrow ep)$
5. Backward-Forward (B-F): $(Y \rightarrow r_1 \rightarrow s) \wedge (Y \rightarrow r_2 \rightarrow ep)$

6. Backward-Backward (B-B): $(Y \rightarrow r_1 \rightarrow s) \wedge (ep \rightarrow r_2 \rightarrow Y)$

where $s \in S, r_1, r_2 \in R, e \in V, Y = \{y_1, y_2, ..., y_l : y_i \in V\}$. A path that matches a given template is deemed valid if the endpoint ep is reachable from all seed elements using the specified relations (r_1, r_2), with the only allowed variation being among the elements of set Y. An example of the pattern `F-F` is shown in the Figure 1. All seed nodes can reach the endpoint by taking the same relations, differing only in the nodes Y, which are the different Grand Prix races.

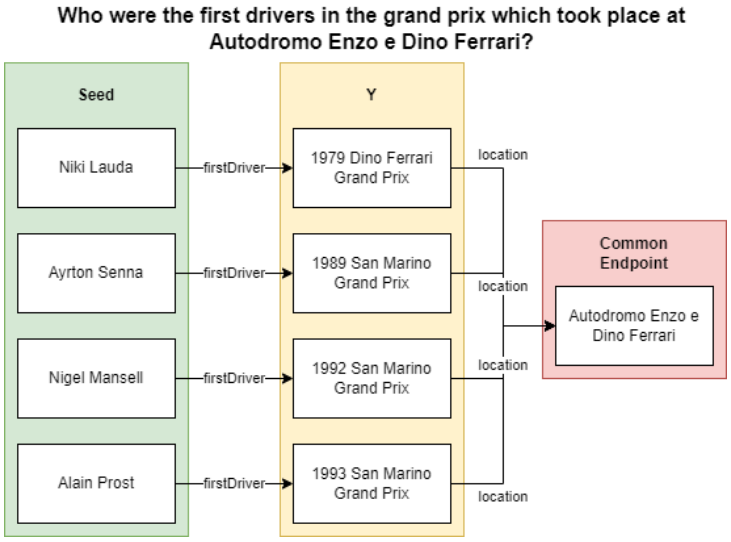

Figure 1: Example of F-F pattern for path: $s \rightarrow$ firstDriver $\rightarrow Y \rightarrow$ location $\rightarrow e$

2.2.1. Algorithm for matching patterns of length one

For all $s \in S$ we query which relation takes us to which endpoint. For example (`Niki Lauda` $\rightarrow$ `type` $\rightarrow$ `Person`). We set a counter for how many times we have seen this relation-endpoint pair. If we iterate over all seed elements and select all elements with a counter value of $|S|$ (seed size), this would include all 1-length paths connecting the seed nodes to a common endpoint.

2.2.2. Algorithm for matching patterns of length two

Finding a common endpoint without exploding the search field is one of the biggest challenges. For example (? $\rightarrow$ `birthPlace` $\rightarrow$ `United States`) would result in a set of millions of entries. Our goal is to keep the number of items in memory to a minimum, and to discard as many items as possible.. Figure 2 demonstrates how the algorithm works. We start by building the path from the seed nodes, since the endpoints are unknown at this point. As a first step, we select all the relations that have connections to all the seed nodes. We call these common relations. Next, we iterate through the common relations one by one and in a nested loop, the seed nodes to query the possible Y sets while limiting the size of the results (for each seed node-relation pair) to 2000 to avoid million-sized sets. The term 'Truncated' refers to this limitation in the name of the algorithm. We call this set the inside points. In the figure this can be any of the colored ellipses when a common relation is selected. From the inside points, we do the same as in the length one patterns, building the counter object, but the same seed node can only increment one to the same counter record. This is an indicator that the endpoint can be reached from the seed node. The key of the counter record is the relation-endpoint pair. If we iterate through all the seed nodes for a single common relation, we can select all the records in the counter object that have a value of $|S|$ (seed size). In Figure 2 the intersection of the colored ellipses shows a path that connects all seed nodes to the same endpoint e1 in a green box via different inside points. Since we know which common relation was chosen we can construct the valid paths e.g. F-F: $s \rightarrow$ `firstDriver` $\rightarrow Y \rightarrow$ `location` $\rightarrow$ `Autodromo Enzo`. The counter object is emptied for the next common relation. For the other patterns to work, only the relation directions had to be reversed, so instead of querying

the tails, we query the heads. Due to the truncation step we might discard nodes that would provide additional elements' endpoints that all the seed nodes connect to, but in practice the algorithm already finds plenty of possibilities to choose from. The pseudocode of the algorithm is given in Algorithm 1.

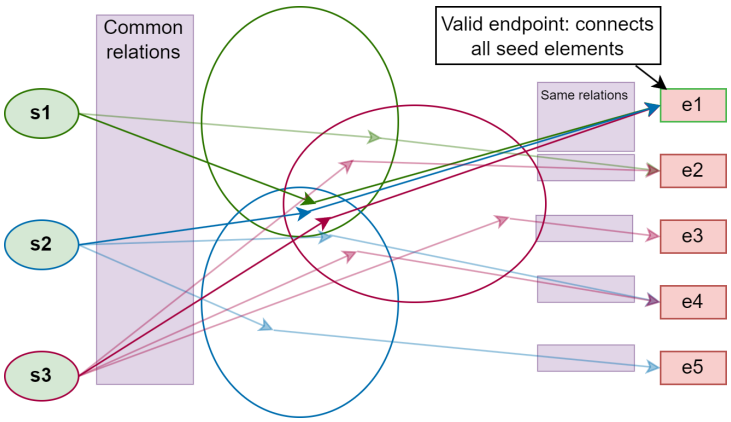

Figure 2: Example of F-F pattern search. Shows possible paths with transparent arrows and a correct path with solid arrows. The colored ellipses represent the set of inside points that a seed node can reach through a common relation.

Algorithm 1 Pattern Length two Matcher

```
1:  procedure F-F(S)                                          ▷ Where S - seed set
2:      cr ← List                                             ▷ common relations container
3:      p ← List                                              ▷ output container
4:      for s in S do
5:          r = query_relations_from (s)                      ▷ Returns all relations starting from s
6:          if cr is empty then
7:              cr = r
8:          else
9:              cr = intersection(cr, r)
10:     for r_1 in cr do
11:         for s in S do
12:             ips = query_tails(s, r_1)                     ▷ Inside points
13:             ips = filters(ips)                            ▷ Limiting the result set
14:             co ← Dictionary
15:             for ip in ips do
16:                 r_2s = query_relations_from(ip)
17:                 for r_2 in r_2s do
18:                     epcs = query_tails(ip, r_2)           ▷ endpoint candidates
19:                     for epc in epcs do
20:                         co.increment_if_new(s, r_2, epc)  ▷ increments the counter of r2 + pep if
    haven't been encountered from seed s
21:             co2 = filter(co, size(S))
22:             for r_2, epc in co2.keys() do
23:                 p.add([?s, r_1, Y, r_2, epc])
24:     return p
```

It is possible to define templates of length three, but the size of DBPedia would most definitely explode the search space making it infeasible.

By running this algorithm the output will contain multiple paths connecting the seed nodes to an endpoint. To list additional entities the seed nodes shall be mask out from the paths and run a SPARQL query that would respond with all the nodes that fit the paths. It is a matter of preference whether all

found paths should be unified along an $\wedge$ operator making it a very strict prediction-set. In most cases, this option will result in only the seed nodes matching the query. The other option is to sample from the queries based on a heuristic to get the final answer.

2.2.3. Simulating the human use case

To address the ambiguity problem and enhance interpretability, we require a well-formulated connection representing all entities in the expanded set, reflecting the user's vision. While our method can be applied to any task with human involvement, we simulate the user in this study by using the gold query from the evaluation dataset.

We evaluate the Truncated Graph Walk method in a *human-in-the-loop* setup. In this system, three alternative queries are presented to the user, who selects one based on personal preference. To simplify decision-making, an ideal system should limit the number of alternatives while ensuring maximum coverage of the query space. We propose a heuristic for selecting a small but diverse set of alternatives by ranking predicted query paths based on the size of their result sets. From this ranking, we select three queries and allow the user to choose the one that best fits their reasoning. In our simulations, we assume the user selects the query with the highest F1 score relative to the gold standard.

The heuristic on which we base our query selection is the following:

1. Select the smallest sized query
2. Discard any query with more than 100 results and select the one closest to the average
3. Select the query closest to 100 results, but not more than 100

Although selecting queries based on their graph structure is more intuitive for users, we chose to present options based on result sets for simplicity as both approaches are interchangeable.

2.2.4. Flaws of the method

In cases where the user wants to list items that do not have a connection in the knowledge graph (KG), such as all cities starting with 'B' in DBPedia, the algorithm cannot reconstruct the full set. While the nodes exist in the KG, the required connections are missing. However, since all our seed and gold sets are based on DBPedia queries, this issue is not a factor in our dataset.

2.3. Textual semantics-based ESE

A baseline solution, we compare the proposed truncated graph walk method, is to expand the seed set based on textual semantic similarities. We transform the texts into a semantic vector space using an embedder, then all operations are performed in the vector space. We utilize the sentenceBert model *all-MiniLM-L6-v2* [9] to obtain the embeddings for each entity of DBPedia using the string representation of the nodes, which adds up to 7 million embedding vectors.

2.3.1. Embedding

Given a seed set, we assume that the correct expansion elements should be close to the seed elements in the vector space. We take the mean of the seed vectors (we call it *center vector*). Imagine a sphere with the center as the center vector and the radius as the Euclidean distance between the center vector and the most distant seed vector. The ESE prediction of this algorithm is the set of all DBPedia nodes whose vector falls within this sphere.

2.3.2. STDev-Embedding

We also consider the assumption that the dimensions of the vector space should have different importance values when calculating the distances for different input seed sets. For example, in the case of `car manufacturers` versus `german brands` the importance of various dimensions in the vector space shall

differ since a seed set can fit into both groups, but the whole expanded sets have a slightly different semantic meaning. To assign weights to the dimensions, we use the standard deviation of each dimension over the seed vectors. Sorting by this value, we select the least spread out N dimensions and discard all others. Based on our assumption the seed elements shall already be close in the vector space, therefore other expansion elements shall align along these dimensions as well. We apply the `Embedding` approach to ESE as described above over this new embedding space.

2.3.3. Simulating human use case

We propose a human-in-the-loop approach for the embedding-based ESE method, where the user labels a small set of entities as correct/incorrect expansions to modify the dimensional weights.

First, we compute the center vector from the seed entities. The user then labels the 5 closest unlabeled entities to the center vector (from all DBPedia entities) using weighted Euclidean distance, with initial weights set to 1. We set up a machine learning loop to adjust the Euclidean distance weights, where seed and positively labeled entities are considered correct, and negatively labeled entities are incorrect. We generate triples in the form of [center, correct, incorrect] from the labeled set. For each triple, the model computes the weighted Euclidean distance between the center-correct and center-incorrect pairs, then calculates the triplet loss. We use the Adam optimizer [10] in PyTorch with a learning rate of 0.75. The weights are constrained within `[0,1]`. The labeling and weight training loop is repeated three times. Finally, we use the final weights to select the most distant correct entity from the center vector and expand the correct set with all DBPedia entities closer than this entity. A high learning rate ensures weights can reach zero if a dimension is deemed unnecessary. For experimentation, we simulate human labeling with an oracle indicating if the entity is part of the gold standard set.

2.4. Large Language Models for the ESE task

Generative Large Language Models (LLMs) like GPT-4 have recently gained popularity. We compare our truncated graph walk method to ChatGPT by prompting GPT-4o (2024-08-06) to evaluate its performance on the ESE task. We conducted two few-shot learning experiments under the assumption that DBpedia was part of GPT-4o's training data, without providing any additional information about the knowledge graph.

In the first experiment, we asked GPT-4 to generate possible explanations for connections between seed elements using DBpedia paths. During testing, GPT-4 occasionally refused to provide answers, citing a lack of database access. Its responses included only 5-6 triples, but the accuracy of gold-standard connections improved as more triples were generated. To improve the model's performance, we refined the prompt iteratively. The final prompt is available in Appendix A. In the second experiment, our goal was to expand the seed set by generating related entities. We included a clarifying question in the prompt to help GPT-4 identify the relationships between the seed elements. The prompt is available in Appendix B.

While the model's structured JSON responses are parsable, there were inconsistencies in formatting (e.g., variations/abbreviations in entity names). To automate evaluation, we discarded the URL component of each entity and used the Levenshtein distance to find the closest match in the gold standard. Entities were considered equivalent if the Levenshtein similarity exceeded 80%. This process was also applied to matching the connection triples. However, this method may lead to false positives or false negatives. [1]

The implementation for dataset creation and methods is available on GitHub: https://github.com/kiscsonti/ContinueTheList

[1] We set the threshold to 80% based on manual investigations and to ensure roughly equal number of false positives and negatives.

3. Results and Discussion

3.1. Experimental setup

We utilize the LC-QuAD [8] dataset which contains 5000 records of natural language questions with the corresponding SPARQL queries over the DBPedia [7] 2016-04 version as a target KG. By running the SPARQL queries we got the gold set of responses (entites) for each question. From these gold sets we kept all the records that had a multi-element response with more than 8 and less than 100 elements simulating ESE use cases. While querying the DBPedia, we discarded all the non-URL like nodes since DBpedia can be noisy and queries can return random literals like: `'181'`, or both `'Memoir'` and `'http://dbpedia.org/resource/Memoir'`. The final dataset with the filters applied yield 310 {English question, SPARQL query, response set of entities} tuples. We randomly sampled `{4, 6, 8}` sized initial seed sets from each response sets and considered a solution perfect if all the gold responses are predicted by the method starting from a seed set. The main evaluation metrics are Precision-Recall-F1 (PRF) scores between the predicted and gold entity sets.

3.2. Results

Table 1 presents a comparison of the performance of the different algorithms.[2] Among the embedding-based approaches, STDev demonstrated the highest performance therefore in only put those numbers in the table. A comparison of embedding-based approaches is provided in Table 3. The Truncated Graph Walk algorithm outperforms all the other methods. Increasing the seed sample size results in a small gain in the F1 score. The embedding-based methods finished second, even losing performance when the seed size is larger.

The Truncated Graph Walk is on par in speed with the STDev method at around 14 hours, even when using a KG as large as DBPedia. GPT-4o performed the worst on the task while being the fastest method, only taking 1 hour to query the responses.

Sample size	Algorithm	Prec.	Recall	F1
4	TruncatedGraphWalk	0.876	**0.942**	**0.887**
	STDev-Embedding	**0.954**	0.189	0.300
	GPT-4o	0.226	0.229	0.204
6	TruncatedGraphWalk	**0.888**	**0.974**	**0.908**
	STDev-Embedding	0.768	0.285	0.376
8	TruncatedGraphWalk	**0.889**	**0.984**	**0.911**
	STDev-Embedding	0.337	0.387	0.284

Table 1
Results of different approaches over the LC-QuAD dataset.

3.3. Discussion on Truncated Graph Walk

The Truncated Graph Walk method offers the advantage of providing human-readable interpretations of expansions by visualizing or verbalizing the graph paths. This allows us to translate graph paths into queries. We evaluated the algorithm's ability to recognize user queries from the original LC-QuAD dataset. We assessed how many gold queries our method can identify without the human-in-the-loop setup, considering all paths connecting the seed set to an endpoint. The percentage of exact matches, where the method successfully identifies the gold SPARQL query, is shown in the ExactMatch column. We observed that if two queries have the same result set size, their elements completely overlap. For example, the queries:
{Skull Gang →currentMembers →?} and *{Skull Gang →bandMember →?}* produce the same results set. This holds for longer paths as well:
{Y →property/layout →?, Y →ontology/designer →Pininfarina} and *{Y →property/layout →?, Y →ontology/class →Light commercial vehicle}*

[2]We run GPT-4o only with the seed size of 4 because of cost issues.

Since multiple queries can yield the same response entity set, we compared queries based on their response entity sets, not the graph paths. A query was considered an exact match if its node set completely overlapped with the gold set. Using this approach, we found that approximately 1 in 4 (27.9% for a seed size of 4) gold queries were missing from the pool of the found query options. Since the user is only presented with 3 options in the human-in-the-loop setup, the realistic match rate for exact gold queries is even lower. To increase exact matches, combining the identified paths using an *AND* operator is necessary, as increasing the seed size has a minimal effect.

Table 2 summarizes the ranking metrics for exactly correct queries. We excluded rows without an exact gold query match and sorted by result size. The ranking metrics used were Mean Rank (MR), Mean Reciprocal Rank (MRR), and Hits at $1, 5, 10$. Results show that half of the gold queries are ranked first, with this improving to 63% with a seed size of 8. However, increasing the hit size revealed that only 75.5% of gold queries appear in the top 10.

We also compared how similar the queries selected in the human-in-the-loop setup are. By matching their triples to the gold set, we found overlaps of 42%, 46.2%, and 47% for seed sizes 4, 6, and 8, respectively. Despite achieving a higher F1 score, less than half of the gold triples are present in the selected queries, suggesting that very similar node sets can be generated from different queries.

Seed Size	MR ↓	MRR ↑	hits@1 ↑	hits@5 ↑	hits@10 ↑	Exact Match ↑
4	1.842±0.22	0.823	0.509	0.696	0.715	72.1%
6	1.362	0.882	0.591	0.740	0.744	74.7%
8	1.227	0.909	0.631	0.755	0.756	75.6%

Table 2
Ranking results of the GraphWalk approach over the LC-QuAD dataset.

To conclude, our solution successfully reproduced the gold set in 72% of all examples, meaning that among the paths connecting the seed elements, there was one that generated the gold set. Each of these paths can be expressed as a natural language sentence, three of which were presented to the user. Thanks to the readability of these sentences, the user can easily interpret the common factor connecting the seed elements and select the one that best aligns with their own vision (gold set). For example, the path mentioned at the beginning of this section is "Current members of Skull Gang". Since the number of examples presented is limited, it is possible that the path that generates the gold set is missing from the connections shown. However, the users are still able to choose the path that most closely matches their view. This explains the low (40-50%) values observed with the matched triples.

3.4. Discussion on the Embedding-based methods

We report on 3 embedding-based variants. First, we have `No annotation` where all the vector space dimensions are used as is, without any trimming using the Euclidean distance. Second, we have the `TOP100` where the vectors are truncated using the STDev embedding method. Finally, we used the `Iterative` process as explained in 2.3.

The results are summarized in the table 3. The `No Annotation` solution produces on average 100 times more nodes than the gold set, resulting in high recall, but poor precision. When we truncated to the top 100 dimensions, it performed the best of all three. It is surprising that the dimension selection heuristic could outperform the iterative variant, where even additional positive examples could leak out during the annotation steps, however more than 95% of the annotated nodes were negatives. In addition, the metrics vary greatly as the seed size increases. With more samples, we noticed that the results dropped sharply. This can be explained by the vectors becoming less distinguishable (more noisy) by their individual dimensions (e.g.: standard deviation). We can see the biggest drop with the iterative approach.

As for the running times, the iterative process took the longest. This is due to the constant search for the nearest neighbors during the annotation loops. Since the vector space is constantly changing due to the learned weights, the use of VectorDBs is not an option, as their structures have to be rebuilt in each

iteration, which has a huge overhead.

SeedSize	Algorithm	Prec.	Recall	F1	Runtime
4	Iterative	0.795	0.201	0.286	7 days 23h
	No Annotation	0.007	**0.562**	0.010	14h
	STDev - Top100	**0.954**	0.189	**0.300**	14h
6	Iterative	0.364	0.306	0.232	7 days 23h
	No Annotation	0.007	**0.562**	0.010	14h
	STDev - Top100	**0.768**	0.285	**0.376**	14h
8	Iterative	0.083	0.429	0.084	7 days 17h
	No Annotation	0.002	**0.732**	0.003	14h
	STDev - Top100	**0.337**	0.387	**0.284**	14h

Table 3
Embedding method results with {Iterative Annotation, No annotation, TopN dimension selection}.

3.5. Discussion on LLM for ESE

For prompt evaluations, we utilized only the seed set consisting of four elements. In the explanation generation task, we were able to match 25.1% of the gold triples to at least one response triple, which is directly comparable to the 42% match rate of the Truncated Graph Walk (see section 3.3). This does not necessarily imply that GPT-4o lacks knowledge about the gold connections, but rather that it is challenging to extract the desired information. To gain deeper insight into the model's responses, we manually examined some outputs and found that the model tends to prioritize more general or common connections over specific ones. Since the gold connections in the dataset are relatively specific, this preference likely explains the observed performance.

In the element generation task, the model's performance was the lowest of the three methods tested, even though the input prompts explicitly specified the exact connection between elements. A major factor contributing to this performance was the model's output limitation in terms of the number of entities generated. The model typically produced between 10 and 30 entities, a limitation that we were unable to overcome through prompt engineering. Furthermore, the entities listed were not always accurate. For example, when asked to list players from the 2016 NBA team, the Phoenix Suns, the model occasionally included current players who were not on the team at the time or players who had previously been on the team but in the most cases the response entities matched the desired group in some aspects, but not the correct one. In appendix C, we provided an example over the three methods.

4. Related work

4.1. Knowledge Graph-based ESE

The first KG-based ESE solution, SEED [11], uses an earlier version of DBpedia (half the size of ours). SEED addresses KG incompleteness by identifying relaxed paths where not all seed elements need to be connected. It employs a probabilistic approach for path discovery and ranks entities for expansion. In contrast, our method directly ranks common aspects and involves a human-in-the-loop annotation process, requiring all seed elements to be present in identified paths. MetaPath [12] also builds trees from seed nodes and expands them along KG edges, ranking elements based on their frequency along these paths, similar to SEED. CoMeSe++ [13] combines KG random walks with textual embeddings from word2vec to expand entity sets. It uses YAGO and DBPedia datasets, but we were unable to obtain the implementation for comparison.

4.2. Corpus-based ESE

Corpus-based ESE has traditionally relied on web data, with early approaches like Google Sets [14] and SEAL [15] using textual sources. More recent methods use subsets of Wikipedia and the APR dataset. SetExpan [1] introduces 40 queries and 41,000 entities, while our dataset includes 310 queries and over 7 million entities. SetExpander [16, 17] uses classifiers with static textual embeddings to predict entities belonging to the seed set. However, it struggles with multi-word expressions, unlike our sentence-transformers, which handle such cases better. Modern approaches like FUSE [18] address ambiguity within seed sets by using BERT's Masked Language Modeling task and skip-gram models for predicting new elements. FGExpan [19] leverages a three-layered taxonomy structure as an auxiliary resource to identify the types of seed entities. By employing multiple scoring techniques, including an MLM and an NLI task, it expands the input set by associating entity types with corresponding elements in the taxonomy. This approach is quite promising in our regard, but effectiveness is constrained by the taxonomy's limited size, consisting of only 28 classes.

4.3. Other Related Works

Corpus-independent methods leverage pre-trained large language models, bypassing the need for external corpora. For example, CGExpan [20] uses a fill-in-the-blank task to identify and rank potential connections between seed elements. GenExpan [21] uses GPT-2 for ESE, expanding sets through multiple fill-in-the-blank tasks, with an approach different from our GPT-4o workflow, which offers token probability distributions for expansions.

The primary focus of the most popular ESE datasets is on named entities. Gajbhiye et al. [22] created a dataset based on ConceptNet and ChatGPT, focusing on more general concepts like `advertisements` or `accomodation`. These approaches address domain variety, unlike previous benchmarks based on Wiki/News data. Our LC-QuAD-based dataset also emphasizes named entities, and expanding the Question to SPARQL dataset could address domain variety in KG-ESE.

MArBLE [23] highlights the variety of available models and notes that their performance varies depending on the seed set. It employs a **human-in-the-loop setup** to assess which model best represents the current expansion set and incorporates suggestions from that model. As sentence-transformers are also numerous there is the possibility to fuse their method with our embedding based solution, however it would result in much higher run times.

Finally, similar tasks like MetaQA [24], ComplexWebQuestions [25], and QALD [26] could serve as alternatives for datasets similar to ours.

No prior work was identified that compares generative models with KG-based ESE solutions.

5. Conclusion

In this paper, we proposed a Truncated Graph Walk algorithm for the Knowledge Graph-based Entity Set Expansion problem. To evaluate the methods for this task, we modified LC-QuAD, a Knowledge Graph Question Answering (KGQA) dataset, and comparatively evaluated three distinct approaches based on different underlying principles. The proposed graph walk-based algorithm uses templates to identify common paths in the graph that connect all seed nodes to a shared endpoint. A second approach involved embedding the KG entities in a semantic vector space using sentence transformers, followed by experiments with dimension selection and iterative weighting techniques. The third approach leveraged few-shot learning through GPT-4o prompting. All approaches were designed to facilitate the integration of human input into the decision-making process, ensuring a common basis for evaluation by simulating human behavior.

The experimental results show that the graph walk-based method outperformed the other approaches, additionally providing explanations in the form of graph paths that are interpretable by humans. In contrast, GPT-4o struggled to identify the correct connections between the seed nodes, as shown by

response elements that were semantically close but not entirely accurate. GPT-4o also had difficulty finding the correct connections (i.e. quasi-explanations) between the seed nodes.

For future directions we plan on applying these methods to other datasets and evaluating a KG-RAG based GenAI solution.

Acknowledgments

This research has been supported by the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: Grammar and spelling check. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] J. Shen, Z. Wu, D. Lei, J. Shang, X. Ren, J. Han, Setexpan: Corpus-based set expansion via context feature selection and rank ensemble, in: Machine Learning and Knowledge Discovery in Databases, Springer International Publishing, 2017, pp. 288–304.

[2] J. Shen, Z. Wu, D. Lei, C. Zhang, X. Ren, M. T. Vanni, B. M. Sadler, J. Han, Hiexpan: Task-guided taxonomy construction by hierarchical tree expansion, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '18, Association for Computing Machinery, 2018, p. 2180–2189. doi:10.1145/3219819.3220115.

[3] J. Shen, J. Xiao, X. He, J. Shang, S. Sinha, J. Han, Entity set search of scientific literature: An unsupervised ranking approach, in: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR '18, Association for Computing Machinery, 2018, p. 565–574. doi:10.1145/3209978.3210055.

[4] P. P. Ray, Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope, Internet of Things and Cyber-Physical Systems 3 (2023) 121–154. doi:https://doi.org/10.1016/j.iotcps.2023.04.003.

[5] O. Yoran, T. Wolfson, B. Bogin, U. Katz, D. Deutch, J. Berant, Answering questions by meta-reasoning over multiple chains of thought, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2023, pp. 5942–5966. doi:10.18653/v1/2023.emnlp-main.364.

[6] R. Logan, N. F. Liu, M. E. Peters, M. Gardner, S. Singh, Barack's wife hillary: Using knowledge graphs for fact-aware language modeling, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2019, pp. 5962–5971. doi:10.18653/v1/P19-1598.

[7] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives, Dbpedia: a nucleus for a web of open data, in: Proceedings of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference, Springer-Verlag, 2007, p. 722–735.

[8] P. Trivedi, G. Maheshwari, M. Dubey, J. Lehmann, Lc-quad: A corpus for complex question answering over knowledge graphs, in: The Semantic Web – ISWC 2017, Springer International Publishing, 2017, pp. 210–218.

[9] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, CoRR abs/1908.10084 (2019). URL: http://arxiv.org/abs/1908.10084.

[10] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, CoRR abs/1412.6980 (2014). URL: https://api.semanticscholar.org/CorpusID:6628106.

[11] J. Chen, Y. Chen, X. Zhang, X. Du, K. Wang, J.-R. Wen, Entity set expansion with semantic features of knowledge graphs, Journal of Web Semantics 52-53 (2018) 33–44. doi:https://doi.org/10.1016/j.websem.2018.09.001.

[12] Y. Zheng, C. Shi, X. Cao, X. Li, B. Wu, Entity set expansion with meta path in knowledge graph, in: Advances in Knowledge Discovery and Data Mining, Springer International Publishing, 2017, pp. 317–329.

[13] C. Shi, J. Ding, X. Cao, L. Hu, B. Wu, X. Li, Entity set expansion in knowledge graph: a heterogeneous information network perspective, Frontiers of Computer Science 15 (2020) 151307. doi:10.1007/s11704-020-9240-8.

[14] S. Tong, J. Dean, System and methods for automatically creating lists, 2008.

[15] R. C. Wang, W. W. Cohen, Language-independent set expansion of named entities using the web, in: Seventh IEEE International Conference on Data Mining (ICDM 2007), 2007, pp. 342–350. doi:10.1109/ICDM.2007.104.

[16] J. Mamou, O. Pereg, M. Wasserblat, I. Dagan, Y. Goldberg, A. Eirew, Y. Green, S. Guskin, P. Izsak, D. Korat, SetExpander: End-to-end term set expansion based on multi-context term embeddings, in: Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations, Association for Computational Linguistics, 2018, pp. 58–62. URL: https://aclanthology.org/C18-2013.

[17] J. Mamou, O. Pereg, M. Wasserblat, A. Eirew, Y. Green, S. Guskin, P. Izsak, D. Korat, Term set expansion based NLP architect by Intel AI lab, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Association for Computational Linguistics, 2018, pp. 19–24. URL: https://aclanthology.org/D18-2004. doi:10.18653/v1/D18-2004.

[18] W. Zhu, H. Gong, J. Shen, C. Zhang, J. Shang, S. Bhat, J. Han, FUSE: multi-faceted set expansion by coherent clustering of skip-grams, CoRR abs/1910.04345 (2019). URL: http://arxiv.org/abs/1910.04345.

[19] J. Xiao, M. Elkaref, N. Herr, G. D. Mel, J. Han, Taxonomy-Guided Fine-Grained Entity Set Expansion, ????, pp. 631–639. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611977653.ch71. doi:10.1137/1.9781611977653.ch71.

[20] Y. Zhang, J. Shen, J. Shang, J. Han, Empower entity set expansion via language model probing, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online, 2020, pp. 8151–8160. doi:10.18653/v1/2020.acl-main.725.

[21] S. Huang, S. Ma, Y. Li, Y. Li, Y. Jiang, H.-T. Zheng, Y. Shen, From retrieval to generation: Efficient and effective entity set expansion, 2023. arXiv:2304.03531.

[22] A. Gajbhiye, Z. Bouraoui, N. Li, U. Chatterjee, L. Espinosa-Anke, S. Schockaert, What do deck chairs and sun hats have in common? uncovering shared properties in large concept vocabularies, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, 2023, pp. 10587–10596. URL: https://aclanthology.org/2023.emnlp-main.654. doi:10.18653/v1/2023.emnlp-main.654.

[23] M. Wahed, D. Gruhl, I. Lourentzou, Marble: Hierarchical multi-armed bandits for human-in-the-loop set expansion, in: Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM '23, Association for Computing Machinery, 2023, p. 4857–4863. URL: https://doi.org/10.1145/3583780.3615485. doi:10.1145/3583780.3615485.

[24] T. Gao, P. Fodor, M. Kifer, Querying knowledge via multi-hop english questions, CoRR abs/1907.08176 (2019). URL: http://arxiv.org/abs/1907.08176.

[25] A. Talmor, J. Berant, The web as a knowledge-base for answering complex questions, in: North American Chapter of the Association for Computational Linguistics, 2018. URL: https://api.semanticscholar.org/CorpusID:3986974.

[26] A. Perevalov, D. Diefenbach, R. Usbeck, A. Both, Qald-9-plus: A multilingual dataset for question answering over dbpedia and wikidata translated by native speakers, in: 2022 IEEE 16th International Conference on Semantic Computing (ICSC), 2022, pp. 229–234. doi:10.1109/ICSC52841.2022.00045.

A. Connection listing prompt

```
Given a seed set of 4 nodes from a DBpedia 2016 dump, list all the
    connections that connect the seed elements so that it can be used
     to list additional similar nodes. Even though you do not have
    access to the database, try to answer with a formatted JSON
    listing the triplets and nothing else! We list some examples with
     only a few connections but list as many connections as you can.
    Make the list at least 20 long!
<EXAMPLE 1>
Seed elements: <<ELEMENTS>>
JSON: <<TRIPLES>>
</EXAMPLE 1>
<EXAMPLE 2>
Seed elements: <<ELEMENTS>>
JSON: <<TRIPLES>>
</EXAMPLE 2>
Seed elements:
```

B. Entity listing prompt

```
TASK: List all the entities that fulfill the stated question as if
    the year was 2016! To help you in the task we provide 4 example
    entities that are correct answers from DBPedia. We can also
    guarantee that the number of correct entities are between 8 and
    100 and all of them are present in DBPedia 2016 version. Answer
    in a JSON list format with all the entities that are correct
    answers for the question!
QUESTION: <<QUESTION>>
EXAMPLES: <<SEED_SET>>
JSON:
```

C. Examples

Seed	Gold	Truncated Graph Walk	STDev-Embedding	GPT-4o
Livewire_(DC_Comics)	Batman:_The_Animated_Series	Batman:_The_Animated_Series	The_New_Batman_Adventures	Andrea_Beaumont
Roland_Daggett	Batman_(Terry_McGinnis)	Batman_(Terry_McGinnis)	Livewire_(DC_Comics)	Clock_King_(comics)
Roxy_Rocket	Condiment_King	Condiment_King	Roland_Daggett	Creeper_(comics)
The_New_Batman_Adventures	Derek_Powers	Derek_Powers	Roxy_Rocket	Curare_(comics)
	Freakazoid!	Freakazoid!	The_Batman_Adventures	Harley_Quinn ⊕
	Harley_Quinn	Harley_Quinn		Inque
	Justice_League_(TV_series)	Justice_League_(TV_series)		Livewire_(DC_Comics) ⊕
	Mercy_Graves	Mercy_Graves		Lock-Up
	Ms._Gsptlsnz	Ms._Gsptlsnz		Luminus
	Nora_Fries	Nora_Fries		Phantasm_(comics)
				Red_Claw
				Red_Hood_(Batman_Beyond)
				Roland_Daggett ⊕
				Roxy_Rocket ⊕
				Shriek_(Batman_Beyond)
				Spellbinder_(comics)
				Supergirl_(Kara_In-Ze)
				Terry_McGinnis ⊕
				The_New_Batman_Adventures ⊕

Table 4
Example of the three methods for the question "`Name some comic characters created by Bruce Timm?`". We use ⊕to indicate positive elements for GPT-4o.

Gold path		
?uri creator Bruce_Timm		
TruncatedGraphWalk		
?x type Person	?x creator Bruce_Timm	
GPT-4o		
?x appearance The_New_Batman_Adventures	Roland_Daggett enemyOf ?seed	?x alliedTo ?seed
?x affiliation Roxy_Rocket	Livewire_(DC_Comics) allyOf ?x	?x createdBy Bruce_Timm
?x villainIn Livewire_(DC_Comics)	?x appearance Roland_Daggett	?x appearsInUniverse DC_Universe
?x rival ?seed	?x similarTo ?seed	?x powers telekinesis
?x voiceActor Mark_Hamill	?x teamAffiliation Gotham_City_Crime_Syndicate	?x creator Paul_Dini
Batman enemyOf ?x	?x portrayedBy Kevin_Conroy	?x influencedBy Superman
Bruce_Wayne hasSecretIdentity ?x	?x type FictionalCharacter	

Table 5
Example of founds paths by the TruncatedGraphWalk and GPT-4o for the example in table **??**. All the returned paths are shown in this examples.