ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΌΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Πτυχιακή Εργασία

« White Rabbit: Απόδραση από την κουνελότρυπα των Γνωσιακών Γράφων χρησιμοποιώντας ενσωματώσεις »

Παναγιώτης Αντιβάσης

ΑΜ: ΜΤΠ327

Τριμελής Εξεταστική Επιτροπή

Νικόλαος Παπαδάκης

Χαρίδημος Κονδυλάκης

Σωτήριος Μπατσάκης

ΗΡΑΚΛΕΙΟ, Σεπτέμβρης 2025

HELLENIC MEDITERRANEAN UNIVERSITY

SCHOOL OF HEALTH SCIENCES

DEPARTMENT OF Electrical and Computer Engineering

THESIS

for the Undergraduate Degree

« White Rabbit:

Escaping the Online KG Rabbit Hole Using Embeddings »

«Panagiotis Antivasis»

YD: mtp327

Three-member Examination Committee

Nikolaos Papadakis

Haridimos Kondylakis

Sotirios Mpatsakis

HERAKLION September 2025

Υπέθυνη Δήλωση Συγγραφεά:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον.

Αποδέχομαι ότι η Βιβλιοθήκη μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από την ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο, καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

«Ευχαριστίες ή Αφιέρωση»

Περίληψη

Η εξάπλωση των Knowledge Graphs (KGs) τα τελευταία χρόνια έχει οδηγήσει στη δημιουργία τεράστιων συνόλων δεδομένων που είναι πλέον διαθέσιμα στο διαδίκτυο. Η εξερεύνηση αυτών των γραφημάτων για τον εντοπισμό ουσιαστικών συνδέσεων μεταξύ οντοτήτων αποτελεί μια πολύτιμη, αλλά και απαιτητική διαδικασία. Αυτό οφείλεται κυρίως στο μεγάλο μέγεθος, την πολυπλοκότητα, καθώς και τις περιορισμένες διεπαφές (όπως τα SPARQL endpoints) που προσφέρουν για διαδικτυακή εξερεύνηση. Η προτεινόμενη εργασία εστιάζει στην ανακάλυψη υψηλής ποιότητας διαδρομών μεταξύ δύο οντοτήτων σε online KGs, αξιοποιώντας embeddings (ενσωματώσεις). Αρχικά, θα παρουσιαστεί το πρόβλημα της αναζήτησης διαδρομών με επίγνωση του συμφραζομένου (context-aware path finding), το οποίο οδηγεί σε συνεκτικές διαδρομές που περιλαμβάνουν ιδιαίτερα σχετικές οντότητες. Στη συνέχεια, η προσοχή θα στραφεί στη μέθοδο White Rabbit, μια προσέγγιση που περιλαμβάνει την αξιολόγηση των γειτονικών οντοτήτων μέσω embeddings (ενσωματώσεων), την προτεραιοποίηση της εξερεύνησης με μηχανισμό ουράς, και τη σταδιακή βελτίωση της διαδικασίας αναζήτησης. Η προσέγγιση αυτή θα συγκριθεί με βασικές μεθόδους, όπως δομικές τεχνικές, διάφορες προκαταρτισμένες μεθόδους embeddings (pretrained embedding methods), καθώς και ένα μοντέλο μεγάλων γλωσσών (large language model oracle), αναδεικνύοντας τα πλεονεκτήματα της προτεινόμενης μεθόδου τόσο ως προς την αποδοτικότητα της διαδικασίας όσο και ως προς την ποιότητα των διαδρομών που ανακτώνται.

Λέξεις – Κλειδιά

Γράφοι γνώσης (Knowledge Graphs), εύρεση διαδρομών (Pathfinding), εξερεύνηση γράφων (Graph Exploration)

Abstract

The proliferation of Knowledge Graphs (KGs) in recent years has resulted in the generation of massive datasets now available online. Exploring these graphs to identify meaningful connections between entities is a valuable, yet challenging task. This is mainly due to the large size, the complexity, and the limited interfaces (i.e., SPARQL endpoints) they offer for their online exploration. The proposed paper will be focused on discovering high-quality paths between two entities in online KGs, using embeddings. First, the problem of context-aware path finding will be introduced, which results into coherent paths including highly relevant entities. Then, focus will be directed on the White Rabbit, an approach that involves scoring entity neighbors using embeddings, prioritizing exploration through a queue-based mechanism, and iteratively refining the search process. This approach will be compared with baselines including structural methods, various pretrained embedding methods, and a large language model oracle, showing the benefits of our approach in optimizing both the efficiency of the task and the quality of the retrieved paths.

Keywords

Knowledge Graphs, Pathfinding, Graph Exploration

Περιεχόμενα

Περίληψη vii

Abstract viii

Περιεχόμενα ix

Κατάλογος Εικόνων / Σχημάτων xi

Κατάλογος Πινάκων xii

Συντομογραφίες & Ακρωνύμια xiii

Εισαγωγή 1

Κεφάλαιο 1: Related Work 5

Κεφάλαιο 2: The White Rabbit 7

2.1 Preliminaries 7

2.2 Problem definition 8

2.3 Problem Hardness 10

2.4 The White Rabbit 11

2.4.1 Fuzzy Pathfinding 14

Κεφάλαιο 3. Evaluation 15

3.1 Results 17

3.1.1 Performance of the White Rabbit on the various datasets 17

3.1.2 Metrics as the length increases 18

3.1.3 Fuzzy Pathfinding 19

3.1.4 Comparison with baselines 19

Συμπεράσματα 23

Βιβλιογραφία 25

Παράρτημα Α: «τίτλος παραρτήματος» 37

Παράρτημα Β: «τίτλος παραρτήματος» 38

Κατάλογος Εικόνων / Σχημάτων

Εικόνα 1-1 Φρούτα και Λαχανικά 2

Σχήμα 1-1 Πυραμίδα Μεσογειακής διατροφής.....3

Κατάλογος Πινάκων

Πίνακας 1-1 Στοιχεία φοιτητών με δικαίωμα εκπόνησης Πτυχιακής Εργασίας 3

Συντομογραφίες & Ακρωνύμια

Ακολουθούν κάποια παραδείγματα:

ΘΕ	Θεματική Ενότητα
ΠΕ	Πτυχιακή Εργασία

Εισαγωγή

Knowledge Graphs (KGs) are structured databases that represent information as a network of entities and their relationships. They are widely used in artificial intelligence, semantic search, and data integration. DBpedia, YAGO, and Wikidata, for example, are well-known KGs storing encyclopedic knowledge, including billions of triples (e.g., Wikidata as of April 2025 includes 16 billion triples) that are continuously updated and enriched, providing also a SPARQL endpoint for their active exploration.

Effectively exploring such KGs can provide valuable insights into relationships between entities, enabling advanced knowledge discovery and reasoning. A key challenge in KG exploration is identifying meaningful paths between two given entities. This requires not only understanding direct connections between sequential nodes, but also evaluating the semantic relevance of the included nodes.

The problem. Traditional approaches, such as structure-based graph traversal (e.g., BFS, DFS, Dijkstra) and path queries, often fail to capture the most contextually relevant paths. In addition, in many cases, those KGs are enormous or are not available in their most recent status. They only provide access through online SPARQL endpoints, which further complicates their exploration.

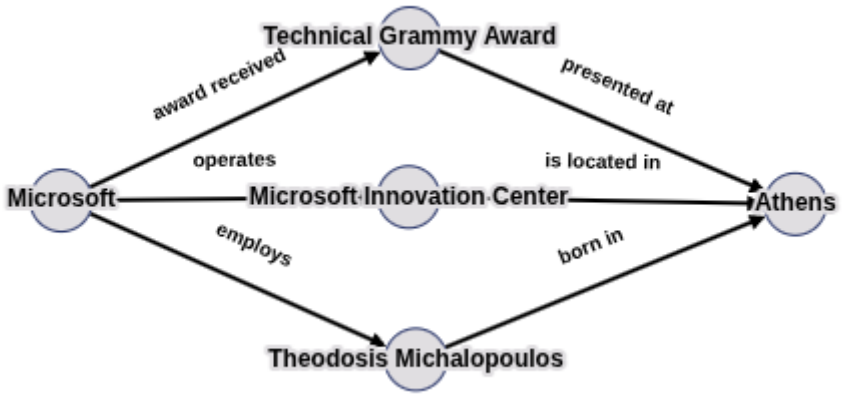

Figure 1: Linking Microsoft with Athens in Wikidata KG.

EXAMPLE 1. *Assume, for example, that we are interested in identifying a path linking the node Microsoft with Athens in the Wikidata KG, shown in Figure 1.*

The SPARQL endpoint offering access to the KG unfortunately cannot accept property path SPARQL queries using wildcards (We could exhaustively sequentially search for paths of different length using the OPTIONAL statement, however, this is still inconvenient and impractical and might not terminate).

As such, we would have to resolve to downloading an existing dump, loading it into main memory, and running a shortest path algorithm like Dijkstra. Unfortunately, Wikidata includes 16 billion statements, so neither loading into main memory 2TB of data can be performed on a commodity computer, nor running the Dijkstra algorithm on a single machine.

Assume, nevertheless, that somehow we manage to identify the following shortest paths:

(1) Microsoft → award received → Technical Grammy Award → presented at → Athens
(2) Microsoft → operates → Microsoft Innovation Center → is located in → Athens
(3) Microsoft → employs → Theodosis Michalopoulos → born in → Athens

Among these paths, the second one offers the strongest semantic similarity between Microsoft and Athens. It represents a direct organizational and operational connection, reflecting Microsoft's active presence in the city through its Innovation Center. The first path shows an award-event coincidence, whereas the third path indicates a personal connection through an employee's birthplace. Although valid, these two paths are more incidental and less indicative of a strategic or organizational link between Microsoft and Athens. To make the problem even worse, imagine that the user searching for a path between Microsoft and the capital of Greece is not sure that the capital is Athens and just provides in text the label Capital of Greece - not the label of an existing node. It is rather common that the users do not know the exact label for both the source and the target nodes, and we would like to be able to give a response in this case too.

To the best of our knowledge, so far, there is no algorithm running on a commodity computer that is able to work directly on online SPARQL endpoints, to provide answers to simple pathfinding queries out of the box, and being able to distinguish the most context-relevant path.

The solution. To address these problems, we introduce the White Rabbit, a novel approach exploiting embeddings to enhance pathfinding by assigning proximity scores to nodes based on the contextual relevance of sequential nodes to reach the target entity. More specifically, our contributions are the following:

(1) **Introduction of the CAP problem.** We are the first to introduce and study the important problem of *Online Context-Aware Pathfinding (OCAP)* between two given nodes, for an online large KG that only exposes its contents through a SPARQL endpoint. The problem, in essence, resolves to *identifying the similarity of all neighboring nodes* in the graph and then, given two nodes, *selecting the path maximizing the total contextual similarity*. Although this problem is central to many other related problems [10, 21], resolving it poses an increased difficulty because of the limitations in accessing the whole graph, its large size, and complexity.

(2) **Addressing the node similarity problem.** Given the prevalence and the high-quality results of the embedding methods for identifying node similarity, in order to resolve the node similarity problem, we exploit pre-trained embeddings that are already available for a number of methods and graphs. As such, the user is not required to download large KGs and re-train them.

(3) **Selecting the path maximizing the total contextual similarity.** Even if we assume that we already have on the graph the similarity between all nodes of the KG, and we can download the KG and load it into main memory, the typical time complexity of Dijkstra and its variations is of O(V+E) * log(V) making the problem infeasible for a large graph using a single computer. Adding the fact that information is available only online through a SPARQL endpoint further complicates the whole setting. In our case, we propose a greedy algorithm for pathfinding between nodes that works directly on the online graph. Our approach follows an iterative strategy: starting from a source entity, the system retrieves its connected nodes and evaluates their relevance to the target node using the \textit{Target Affiliation (TA)} metric. The TA metric is a function that is evaluated using embedding similarity, evaluating the degree to which nodes along the path are affiliated with the target node. The exploration is prioritized based on this metric, continuing until the target entity is reached.

(4) **Fuzzy pathfinding.** We move one step further, allowing users to find paths given only the node to begin with and a label which might not even be available in the data, as the target node. In order to achieve that, we still exploit embeddings,

considering that we reach our target when the embedding of the current node has a high similarity with the embedding of the (perceived) target.

(5) **Rich Evaluation.** In order to evaluate our approach, we present a rich set of experiments using three real online KGs, i.e., Wikidata, DBpedia, and YAGO, and several competitors using embeddings, structural walks using SPARQL queries, and an LLM. We demonstrate the superiority of our approach in terms of both quality and efficiency.

To the best of our knowledge, this is the first approach focusing on context-aware pathfinding on online KGs.

The remainder of this paper is structured as follows: Section 2 presents related work. Then Section 3 introduces preliminaries, defines the problem and presents the corresponding algorithm. Section 4 presents our evaluation and finally Section 5 concludes the paper and presents future work.

Κεφάλαιο 1: Related Work

Pathfinding in KGs. Pathfinding in KGs is a well-explored problem in artificial intelligence and database research. Traditional methods for finding paths between entities include breadth-first search (BFS), depth-first search (DFS), and shortest path algorithms such as Dijkstra's algorithm. These methods, while effective for generic graphs, often struggle with semantic relevance because they do not incorporate meaning or context into their exploration process.

Several studies have attempted to enhance pathfinding by incorporating heuristic-based methods. For example, Path Ranking Algorithms [13] leverage random walks and supervised learning to predict paths between entities. Rule-Based pathfinding [23], involves the use of logical rules to constrain the search space and improve efficiency. These methods improve upon basic traversal algorithms but often require domain-specific knowledge to define relevant path constraints.

Lately, Pathfinder [5, 10] focuses on returning paths from KGs that match regular path queries, however it requires that the user already knows the edges of interest and that the whole graph is available for computation, which in our case do not hold. Further, it completely ignores the context in the exploration process.

SPARQL Query Expansion for Graph Exploration. SPARQL, the standard query language for querying KGs, allows for retrieving structured data from RDF-based knowledge graphs. However, one of the main challenges in using SPARQL for pathfinding is the necessity of predefined query structures. If the connection between two entities is unknown, iterative broadening approaches [4] have been proposed to dynamically construct multi-hop queries, starting with simple queries and progressively expanding them to include more intermediary entities. Other studies [12] have investigated adaptive query expansion techniques that optimize SPARQL queries based on past query results, improving search efficiency, or use reinforcement learning [7] to generate optimal query paths dynamically. These approaches, although they provide a more flexible way to explore knowledge graphs,

still require structured query formulation, limiting accessibility to non-expert users, and completely ignoring the exploration context.

Large Language Models for Knowledge Graph Exploration. Recent advancements in Large Language Models (LLMs) have introduced new possibilities for knowledge graph exploration. LLMs have demonstrated remarkable capabilities in understanding natural language queries, extracting relationships between entities, and even generating structured outputs such as SPARQL queries.

However, exploiting LLMs for pathfinding in KGs is still an emerging research area. Recent studies [20] have proposed using prompt engineering techniques to guide LLMs in scoring entity relevance dynamically, or improving search efficiency on research papers KG by prioritizing nodes that have a higher semantic relevance to a target [6]. However, approaches so far have limited applicability to KG traversal tasks.

Our approach moves beyond the limitations of existing pathfinding approaches in KGs -- which assume that the whole graph is available, the SPARQL query expansion techniques -- that require structured query formulation, and is unique in considering the context in the exploration process. Further, it moves beyond LLMs with limited applicability on generic KG traversal, which might also introduce monetary cost. In our evaluation, we compare with representatives from those domains as well, demonstrating the superiority of our solution in terms of both efficiency and effectiveness.

Κεφάλαιο 2: The White Rabbit

2.1 Preliminaries

A Knowledge Graph (KG) is a structured representation of knowledge that consists of entities and their relationships. In a KG, entities represent real-world concepts, objects, or events, while the relationships capture how these entities are related. In this work, we focus on RDF KGs, as RDF is among the widely used standards for publishing and representing data on the Web. Those datasets are based on triples of the form $(s\ p\ o)$, which record that $subject\ s$ is related to $object\ o$ via property p. Formally, representation of RDF data is based on three disjoint and infinite sets of resources, namely: URIs (U), literals (L), and blank nodes (B). A key concept for RDF is that of URIs or Unique Resource Identifiers; these can be used in either of the s, p, and o positions to uniquely refer to some entity, relationship, or concept. Literals (constants) are also allowed in the o position. Blank nodes in RDF allow representing a form of incomplete information for unknown constants or URIs. As such, a triple is a tuple $(s\ p\ o)$ from $(U \cup B) \times U \times (U \cup L \cup B)$. An RDF dataset can be represented as a labeled directed graph. Given a set A of labels, we denote by $G = (V$, $E)$ an A-edge labeled directed graph whose vertices are V , and whoseedges are $E \subseteq V \times A \times V$. In the graph, nodes represent subjects or objects and labeled directed edges represent properties.

*Definition 3.1 (**Path**)*. A path from $vs \in V$ to $vt \in V$, i.e. $P\ (vs, vt)$, is a finite sequence of edges, which connect a sequence of nodes, starting from the node vs and ending in the node vt.

Moreover, we denote with $|P\ (vs\ ,\ vt\ )|$ the length of the path, i.e., the number of edges in it. Further we denote with P $(vs\ ,\ vt\ )$ the set of all paths from node vs to node vt . Querying. For querying RDF datasets, W3C has proposed SPARQL [1]. Essentially, SPARQL is a graph-matching language. A SPARQL query Q defines a graph pattern P that is matched against an RDF graph G. This is done by replacing the variables in P with elements of G such that the resulting graph is contained in G. The most basic notion in SPARQL is a triple

pattern, i.e., a triple where every part is either an RDF term or a variable. One or more triple patterns form a basic graph pattern (BGP). The result of a BGP is a bag of solution mappings, where a solution mapping is a partial function μ from variable names to RDF terms. On top of these basic patterns, SPARQL also provides more relational-style operators like optional and filter to further process and combine the resulting mappings, negation, property paths, assignments, aggregates, etc.

2.2 Problem definition

Traversing a graph to discover meaningful paths or relationships between two given entities is central to many problems like query answering [21], knowledge base completion [15], recommendation systems [24], link prediction, and entity resolution [11], where understanding the structure and semantics of intermediate nodes and edges enables more accurate and interpretable results.

Identifying meaningful paths. Identifying semantically meaningful connections is crucial for meaningful knowledge discovery. Embedding methods have become essential for identifying similarity between nodes in knowledge graphs (KGs), as they enable the transformation of rich, symbolic structures into continuous vector spaces that preserve latent semantic relationships. Unlike traditional symbolic approaches, which rely on costly path traversals or handcrafted similarity metrics, embedding-based methods offer scalable and learnable alternatives that capture both local and global graph structure. These embeddings allow efficient computation (e.g., via cosine similarity or dot product) and generalize well across tasks, making them crucial for applications ranging from semantic search to recommendation and entity resolution.

Definition 3.2 (Node Semantic Similarity). Let $G = (V , E)$ be a KG. Assume that each node $v \in V$ is associated with a learned embedding vector e$v \in$ Rd . The semantic similarity between two nodes $vs , vt \in V$ is defined as the cosine similarity between their embedding vectors:

$$\text{sim}(v1, v2) = \cos(\text{e}v1 , \text{e}v1) = \text{e}v1 \cdot \text{e}v2 / \|\text{e}v1 \| \cdot \|\text{e}v1 \|$$

where · denotes the dot product and $\| \cdot \|$ denotes the Euclidean norm.

Now, having a way to define semantic similarity between two nodes, we define next the target-affiliation metric for an entire path.

Definition 3.3 (Target Affiliation Metric). Let $G = (V , E)$ be a KG, and $P (vs , vt)$ a path in G between vs and vt . The target-affiliation (TA) metric of $P (vs , vt)$ is the mean similarity of all nodes in the path with the target node vt , i.e.,

$TA(P (vs , vt)) = 1 / |P (vs , vt)| |P (vs ,vt) |\sum i=1 sim(vi, vt) (1)$

This metric evaluates the degree to which nodes along the path are affiliated with the target node. A higher mean value indicates that the path remains contextually close to the target node, ensuring relevance and meaningful connectivity.

Now we are ready to define the online context-aware pathfinding problem.

Definition 3.4 (Online Context-Aware Pathfinding (OCAP)). Let $G = (V, E)$ be a KG allowing access through a SPARQL endpoint. Given two nodes $vs , vt , \in V$, the objective is to find a path $P (vs , vt)$ such that the target affiliation score $T A(P (vs , vt))$ is maximized.

$P* = \arg\max P \in P (vs ,vt) T A(P (vs , vt))$

where $P (vs , vt)$ is the set of all paths from node vs to node vt.

Example 2. Assume that for linking the node Microsoft to Athens, we already have access to the three candidate paths shown in Example 1.

(1) Microsoft → Technical Grammy Award → Athens
(2) Microsoft → Microsoft Innovation Center → Athens
(3) Microsoft → Theodosis Michalopoulos → Athens

Each node is associated with an embedding vector $ev \in Rd$. The semantic similarity of the various nodes with the Athens is shown in the following table:

v	**Συχνότητες**
Microsoft	0.35
Technical Grammy Award	0.20
Microsoft Innovation Center	0.78
Theodosis Michalopoulos	0.45

We compute:

PC($P1$) = 1/2 (0.35 + 0.20) = 0.275

PC($P2$) = 1/2 (0.35 + 0.78) = 0.565

PC($P3$) = 1/2 (0.35 + 0.45) = 0.40

As such, the most context-relevant path is the second one, confirming also our intuition in Example 1 that this the path that offers the strongest semantic similarity between Microsoft and Athens.

2.3 Problem Hardness

It is easy to see that the even the problem of online context-aware pathfinding is an NP-complete problem:

Theorem 3.5. The online context-aware pathfinding problem is NP-complete.

PROOF. (sketch) The problem is a variant of the maximum-weight path problem, where edge weights are derived from similarities between embeddings. Cycles should be avoided, whereas the number of candidate paths between two nodes in a general graph is exponential in the number of nodes. Given that the problem, even without data access restrictions, is already hard, accessing the data through a SPARQL endpoint, i.e. the online version of the problem, makes it even harder.

As such, our capacity to traverse the whole graph becomes rather difficult, whereas effectively training embedding models through online SPARQL endpoints is another problem.

2.4 The White Rabbit

To address these difficulties, we propose the White Rabbit based on the following two key ideas:

(1) **Accessing Data through Online SPARQL endpoints.** Rather than preloading the entire graph into memory, the algorithm follows a gradual exploration strategy, where data is retrieved on demand as the search progresses. Starting from a source node, it issues SPARQL queries to discover immediate neighbors and incrementally expands the search frontier based on a priority queue. This on-the-fly access model enables the algorithm to scale to very large graphs %while minimizing bandwidth and memory usage and also makes it suitable for interacting with live datasets where downloading the full graph is impractical or impossible.

(2) **Calculating Node Similarity.** To guide the traversal toward the target, the algorithm leverages pretrained node embeddings that capture the latent semantic structure of the knowledge graph. During execution, the algorithm computes the similarity between candidate nodes and the target node and prioritizes the exploration of nodes that are semantically closer. This embedding-driven heuristic enables a form of goal-oriented search that is both flexible and generalizable, allowing the algorithm to identify contextually relevant paths even in noisy or incomplete graphs. In our case, we adopted Wikipedia2Vec embeddings [22] pretrained on Wikipedia articles; however, in our evaluation, we also compare with WORD2VEC [8], FASTTEXT [3], and SBERT [17]. We consider the problem of training embeddings through an online SPARQL endpoint orthogonal to our approach, which has already been investigated by a plethora of research works [16], and as such is not further investigated in this paper. The overall algorithm is shown in Algorithm 1 and takes as input the start node, the target node, and a SPARQL endpoint, and returns the path between the two nodes or an empty set if no path is found. In the beginning, the algorithm initializes a priority queue Q, with the source node and the similarity between the first and the last node (line 1). The priority queue prioritizes nodes with higher similarity to the target node. Further, the set of visited nodes is initialized, a variable that will be used to avoid revisiting nodes already in the path (line 2), as well as the path map that stores the path (line 3). The algorithm continues exploring while there are candidate nodes in the priority queue (line 4). Each time, the node with the highest similarity to the target is extracted (line 5). If we have already visited that node, we continue with the next node in Q (lines 6-7). Otherwise, we add the node to the visited ones (line 8). If the node is the target node, then we return the whole path to the user (lines 9-10). If we have not already visited the node and we

have not found the target node, we send a simple SPARQL query to find all neighbors of the current node (line 11), i.e., SELECT * WHERE $\{vc\ ?p\ ?o.\}$. For each one of the neighbors ri (line 12), if we have not seen it before (line 13), we construct the full path to ri by extending the path to vc with ri (line 14). Further, we calculate the similarity between ri and the target node and insert ri into the priority queue.

Algorithm 1: The White Rabbit

Input: Source node v_s, Target node v_t, SPARQL endpoint E
Output: Path P from v_s to v_t or $\emptyset$

1 $Q \leftarrow$ priority queue with $(sim(v_s, v_t), v_s)$
2 $Visited \leftarrow \{\}$ // To avoid revisiting nodes
3 $Paths[v_s] \leftarrow [v_s]$ // Initialize path map
4 **while** Q *is not empty* **do**
5 $(_, v_c) \leftarrow extractMax(Q)$ // Node with highest similarity
6 **if** $v_c \in Visited$ **then**
7 **continue**
8 $Visited \leftarrow Visited \cup \{v_c\}$
9 **if** $v_c == v_t$ **then**
10 **return** $Paths[v_c]$
11 $R \leftarrow executeSelectQuery(v_c, E)$ // Outgoing edges from v_c
12 **foreach** $r_i \in R$ **do**
13 **if** $r_i \notin Visited$ **then**
14 $Paths[r_i] \leftarrow Paths[v_c] + [r_i]$
15 $s \leftarrow sim(r_i, v_t)$
16 $insert(Q, (s, r_i))$
17 **return** $\emptyset$

EXAMPLE 3. We now illustrate the White Rabbit algorithm in our running example, finding the path between Microsoft and Athens. The algorithm begins by initializing the priority queue with the similarity score between Microsoft and Athens:

Q ← [(0.35, Microsoft)]

Paths[Microsoft] ← [Microsoft]

Then, the node with the highest score is dequeued, i.e., (0.35, Microsoft). The algorithm, next, queries the direct neighbors of Microsoft using the following SPARQL query:

SELECT * WHERE { Microsoft ?p ?target }

The query returns the following neighbors:

Technical_Grammy_Award, similarity = 0.20

Microsoft_Innovation_Center, similarity = 0.78

Theodosis_Michalopoulos, similarity = 0.45

Each neighbor is added to the path map.

Paths[Technical_Grammy_Award] ← [Microsoft, Technical_Grammy_Award]
Paths[Microsoft_Innovation_Center] ← [Microsoft, Microsoft_Innovation_Center]
Paths[Theodosis_Michalopoulos] ← [Microsoft, Theodosis_Michalopoulos]

The queue now becomes:

Q ← [(0.78, Microsoft_Innovation_Center), (0.45, Theodosis_Michalopoulos), (0.20, Technical_Grammy_Award)]

Similarly, Microsoft_Innovation_Center is dequeued and its neighbors are retrieved, i.e., Athens with $similarity$ = 1.00. The path is updated and added to the queue:

Paths[Athens] ← [Microsoft, Microsoft_Innovation_Center, Athens]

Then Athens is added first in the queue. Finally, Athens is dequeued. Since it matches the target node, the algorithm terminates and returns the corresponding path.

Complexity. The algorithm resembles a Best-First Search, but over a dynamic graph where edges are fetched via SPARQL. For each node visited, a SPARQL query is executed to retrieve its neighbors, leading to a total of O ($|V| \times TQ$), where TQ is the cost of the query 2. Each discovered neighbor requires the computation of a similarity score with respect to the target node, which contributes O ($|V| \times \kappa$) assuming embeddings of dimensionality κ. The priority queue used to guide exploration based on similarity involves O ($|E| \times log|V|$) operations in the worst case3. Therefore, the total time complexity of the algorithm becomes O ($|V| + |E| \times log|V|$)

2.4.1 Fuzzy Pathfinding

In KGs, it is common that users do not know the labels of the nodes of interest, and often struggle to find them, - many of the users do not even know how to write a SPARQL query. As such, we can slightly change Algorithm 1 so that we don't require the users to know the label of both nodes to be connected, but only one of the nodes is enough.

The corresponding algorithm is shown in Algorihtm 2. The algorithm starts similarly to the Algorithm 2; however, now we provide as input a similarity threshold, which is used to compare the embedding of each node with the embedding of the target node (line 9). Now we don't expect that we will be able to find exactly the target node, but reaching a similarity threshold is enough.

Adding a similarity-based early termination condition does not change the worst-case complexity, but it can significantly reduce the average-case runtime in practice, especially when nodes close to the target in embedding space are discovered early. The search becomes more goal-biased and adaptive, making it suitable for approximate or exploratory search tasks in large and noisy KGs. The algorithm can also be used when we only know the target node, but we are not sure about the source node, by simply asking Algorithm 2 to connect the target (as source) with the source (as target).

EXAMPLE 4. Assume in our running example that a user knows the starting node, i.e., Microsoft; however, she is unsure about the label of the target node, i.e., the Athens. Nevertheless, the user still has relevant information about the target node, i.e., that it is the capital of Greece. We use Wikipedia2Vec to transform the “Capital of Greece” to an embedding, and then we call Algorithm 2. The algorithm is able to identify the correct path.

Algorithm 2: The Fuzzy White Rabbit

Input: Source node v_s, Target node v_t, SPARQL endpoint E, similarity threshold T
Output: Path P from v_s to v_t or $\emptyset$

$Q \leftarrow$ priority queue with $(sim(v_s, v_t), v_s)$
$Visited \leftarrow \{\}$ // To avoid revisiting nodes
$Paths[v_s] \leftarrow [v_s]$ // Initialize path map
while Q *is not empty* **do**
  $(_, v_c) \leftarrow extractMax(Q)$ // Node with highest similarity
  if $v_c \in Visited$ **then**
    continue
  $Visited \leftarrow Visited \cup \{v_c\}$
  if $sim(v_c, v_t) \geq T$ **then**
    return $Paths[v_c]$
  $R \leftarrow executeSelectQuery(v_c, E)$ // Outgoing edges from v_c
  foreach $r_i \in R$ **do**
    if $r_i \notin Visited$ **then**
      $Paths[r_i] \leftarrow Paths[v_c] + [r_i]$
      $s \leftarrow sim(r_i, v_t)$
      $insert(Q, (s, r_i))$
return $\emptyset$

Κεφάλαιο 3. Evaluation

In this section we present the extensive experimental evaluation of our algorithms.

Implementation. All algorithms were implemented using Python and are available online (https://anonymous.4open.science/r/The-White-Rabbit-Algorithm-0F73). In addition, the evaluation was performed using a commodity computer, i.e., Windows 10 with an Intel® CoreTM i3 10100 CPU @ 3.60GHz (4 cores) and 16 GB RAM.

Datasets. Three real-world KGs were used for evaluation: YAGO, DBpedia and Wikidata. YAGO [18] is a KG combining the entities of Wikipedia with WordNet for taxonomy. It currently contains 120 million RDF triples. DBpedia [14] is a multilingual knowledge base that collects formatted and queryable data from Wikipedia. DBpedia has about 9.5 billion RDF triples. Wikidata [19] structures and centralizes all Wikipedia knowledge into a multilingual repository for more efficient querying and re-usage. It is the largest of the three, containing over 16 billion RDF triples.

Baselines. We compare our algorithms with several baselines using various embedding methods, SPARQL query expansion, and LLM-based pathfinding.

Embedding methods. We try several pretrained embeddings, i.e., WORD2VEC [8], FASTTEXT [3] and SBERT [17]. WORD2VEC is foundational in the field of word embedding techniques, converting each word into a vector representation that can then process with vector arithmetic. FASTTEXT enhances the WORD2VEC framework, by converting words to the sum of their n-grams and models their meanings using Gaussian mixture densities. Opposite to its predecessor, FASTTEXT can manage infrequent/slang terms and polysemy, as well as cross-linguistic generalization. However, it requires a lot of computing power as it is much more complex. SBERT extends the pretrained embedding BERT, utilizing a siamese network structure. Despite being efficient in entity similarity and semantic search, it is rather computationally heavy, often requiring GPU support, and performs better with sentences instead of single words.

In *SPARQL query expansion (QE),* we use iterative broadening of the SPARQL query, including only source and target entities as non-variables in an ever-expanding query. In this way, we try to (exhaustively) find the path(s) between the source and target entities.

In the LLM pathfinding method we devised, instead of pretrained embeddings, we use Anthropic Claude (Sonnet 3.7) through appropriate prompts in order to identify the similarity scores between the nodes [2]. We initially explored GPT-4-turbo, however, the level of creativity and improvisation of the model was higher with low-quality results.

To evaluate the various systems, we randomly selected 50 paths from each dataset and ran both the baselines and our algorithms. Due to prolonged processing times, we added a timeout of 360 seconds (6 minutes) on each algorithm, measured after model loading. The presented results are the average of ten executions.

Metrics. For the various algorithms and baselines, we evaluate success rate, efficiency, discovered path length, and quality of the retrieved path. Success rate is the percentage of source-target pairs that successfully reached the pathfinding target before the timeout. For evaluating the quality of the retrieved path, we use Target Affiliation (Definition 3.3) and Path Coherence.

Definition 4.1 (Path Coherence). Let $G = (V , E)$ be a KG, and $P\ (v1, v2)$ a path in G between $v1$ and $v2$. The path coherence (PC) of a $P\ (v1, v2)$ is the mean similarity of the sequential nodes in the path, i.e.

$$PC\ (P\ (v1, v2)) = 1/(n - 1)\ n-1\sum i=1\ sim(vi, vi+1)\ (2)$$

where $n = |P\ (v1, v2)|$.

The Path Coherence metric measures how semantically coherent the hops are between consecutive nodes in the path. A higher value indicates that the transitions between nodes are semantically meaningful.

3.1 Results

3.1.1 Performance of the White Rabbit on the various datasets

For warming up, we present the results for success rate, efficiency, and quality of the White Rabbit in Figure 2. The results reveal notable variations in performance and behavior.

Most importantly, the success rate is highest on Wikidata (87%), followed by DBpedia (75%) and YAGO (45%), indicating that Wikidata is more navigable under the algorithm's heuristic despite its larger size.

In terms of mean execution time for the successful queries, the algorithm is fastest on YAGO (28.4s) as it is the smallest dataset, followed by Wikidata (59.9s), while DBpedia is the slowest at 104.5s, as larger paths should be retrieved. More specifically, DBpedia yields the longest average paths (13.7), while YAGO and Wikidata produce shorter paths (6.9 and 10.5, respectively).

The PC score, which reflects path coherence, follows the same trend: lowest in YAGO (9.0) and highest in DBpedia (29.8), with Wikidata in between (17.9), suggesting a trade-off between coherence and search depth. Lastly, the TA score, a measure of topical alignment between the retrieved node and the target, is highest in Wikidata (0.48), suggesting that its embeddings or structure better preserve semantic proximity.

Overall, Wikidata offers the best balance between efficiency, accuracy, and semantic relevance, whereas DBpedia offers high success at the cost of time and longer paths. YAGO overall presents a more challenging environment for pathfinding despite its relatively small size.

3.1.2 Metrics as the length increases

Figure 3 presents how the Time, PC, and TA behave as the length of the discovered path increases for the various datasets using Linear Regression - for the paths that were successfully calculated without timeout.

As shown for all three datasets, time presents a strong positive correlation with the length of the discovered path. As expected, longer paths require larger pathfinding times. In all three datasets, PC also has a relatively positive trend, albeit less steep than time. This shows that with larger paths, we have the chance to identify sequentially more semantically closer nodes in the whole exploration. Finally, TA shows diversity across the datasets, showing a relatively positive correlation with YAGO and DBpedia and a slight negative correlation with WikiData. We expect than in nodes with more distance from the target node, we will have a smaller similarity with it, and as such we expect that TA indeed will show a smaller increase, whereas WikiData seems to be the most challenging environment for pathfinding on large paths, due to its crowd-sourced nature, inhibiting long paths from being semantically consistent as they grow [9].

3.1.3 Fuzzy Pathfinding

Next, we contrast the White Rabbit algorithm with the Fuzzy White Rabbit for various similarity thresholds (90%, 80%, and 70%). In essence, when the similarity threshold of the Fuzzy White Rabbit algorithm is 100% the algorithm corresponds to the White Rabbit algorithm. The results are shown in Figures 4. As expected, as the similarity threshold is relaxed, the success rate increases, and the pathfinding stops earlier, which results in shorter paths and faster execution times. Of course, as expected as well, this has an impact on the quality of the discovered path, which is reduced as the similarity threshold is relaxed. Notably, although in DBpedia and Wikidata the execution time drops around 44%, the quality of the discovered paths drops only marginally (TC drops by 20% and TA drops by 30%), showing that the Fuzzy White Rabbit can provide a good approximate solution trading almost linearly quality for efficiency.

3.1.4 Comparison with baselines

Figure 5 compares White Rabbit (WR) with five baselines, i.e., SPARQL Query Expansion (QE), LLM pathfinding (LLM), and alternative embedding methods, WORD2VEC, FASTTEXT, and SBERT, in the three datasets.

First and foremost, regarding success rate, we can observe that the White Rabbit has consistently the highest, answering up to 87% of the queries in the WikiData dataset. This demonstrates its robustness and effectiveness in finding paths. WR dominates both EQ and LLM in all datasets, and especially on WikiData, suggesting limitations in their ability to guide pathfinding effectively in more structurally complex or sparse knowledge graphs. SBERT, FASTTEXT, and WORD2VEC achieve moderate to high success rates, with SBERT slightly outperforming the other two in most cases. However, WR still outperforms all other embedding methods.

Regarding average execution time in order to be fair with the comparison, in the figure we include for each query that times out, the time-out time as well. As shown, WR is the slowest embedding method as it produces embedding vectors with larger dimensions; however, as we will see, this has a direct impact on the quality of the results. Further, it is always faster than the LLM method, which requires significant execution time for the LLM to respond. Finally, it is slightly slower than SPARQL query expansion as it needs, on top of query answering, to perform similarity calculations.

The third column of graphs in Figure 5 presents the average path of the retrieved paths per dataset. Note that competitors are able only to retrieve the paths of nodes that are close and, as such, they fail to retrieve longer paths (see the graphs on the success rate). This is the reason that in the figure we see a larger average length for WR, as it is able to find also the connections on significantly larger paths.

Moving to the quality PC and TA, we also see that the quality of the retrieved paths in WR is around 20% better than QE and LLM, demonstrating the high value of our approach, independent of the fact that both QE and LLM are able to retrieve paths for only a subset of the queries. When comparing with other embedding methods, those methods are able to retrieve just a minor subset of the total number of queries, which are usually shorter and have better quality.

In summary, the WR method offers the best overall performance, excelling in success rate and path quality. Other embedding methods return only a fraction of answers (WORD2VEC, SBERT and FASTTEXT). LLM is resource-intensive and prone to producing verbose, less relevant paths. QE, although fast, fails to deliver paths with high quality when compared to WR and fails to succeed as well in success rate.

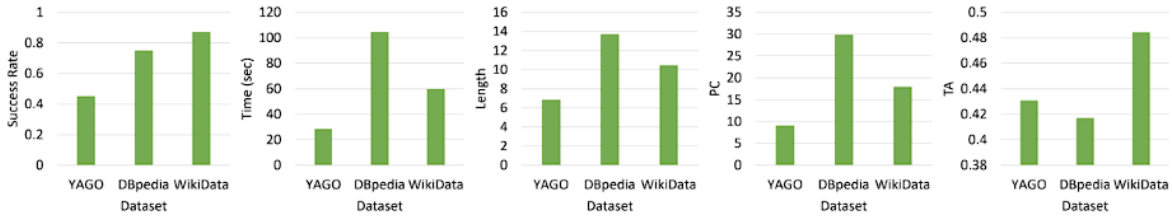

Figure 2: Success rate, Time, Length, PC and TA for all datasets

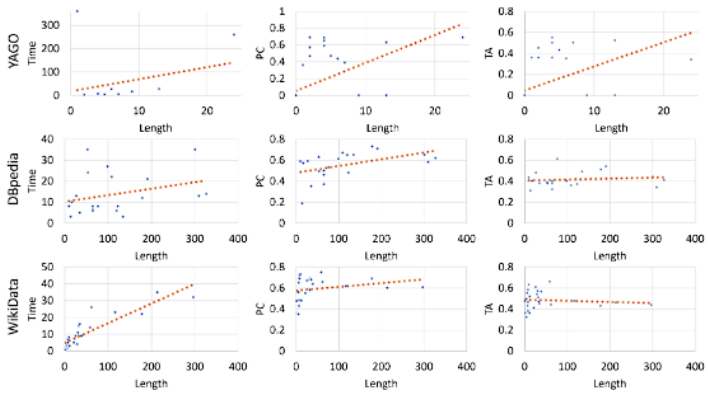

Figure 3: Time, TA and PC as the length increases for all datasets

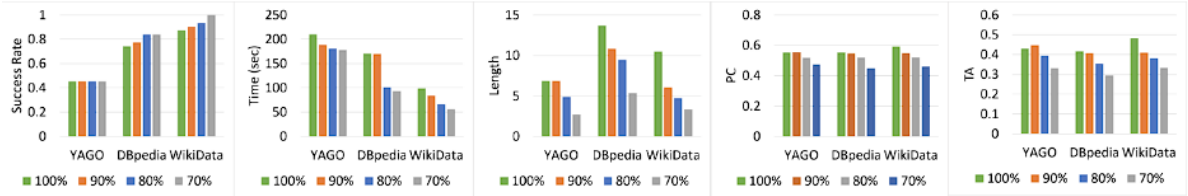

Figure 4: Time, length, PC and TA for the Fuzzy White Rabbit with various similarity thresholds

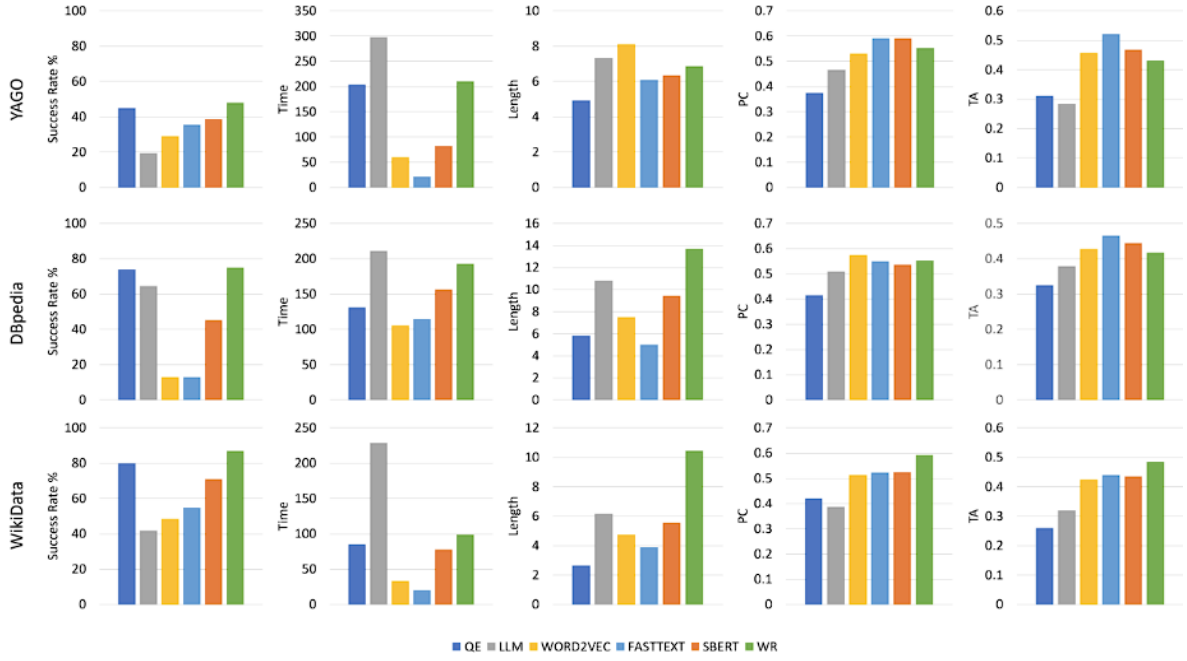

Figure 5: Comparison with competitors

Συμπεράσματα

In this paper, we present the White Rabbit, the first pathfinding algorithm addressing the OCAP problem, being able to run on commodity computers and working directly on online SPARQL endpoints. In addition, we present an approximate algorithm for trading efficiency for quality, allowing us to discover paths even in cases when we only know the exact label of one of the nodes we are looking for. We compare our algorithm with five baselines, showing the advantages of our solution, outperforming all competitors in both efficiency and quality. Further, we show that using the Fuzzy White Rabbit, we can almost linearly exchange efficiency for quality.

Limitations. Our approach uses pretrained embeddings that work well with encyclopedic KGs, but might not work well with other KGs, and transfer learning or custom training might be required for other KGs. However, as already mentioned the embedding method and its training is orthogonal to our approach and a new better method can be plugged-in in seconds.
Future Work. For future work, we intend to explore how to optimize the efficiency of our algorithms by parallelizing the exploration of the priority queue or inserting only the top-k nodes. Further, we intend to test the system in related practical scenarios such as knowledge graph navigation and recommendation systems to assess its effectiveness in such scenarios.

Disclosure of Generative AI Use

This work complies with the ACM Policy on the Use of Generative AI in Publication. Generative AI tools were used as follows:

Language editing and formatting support: ChatGPT (OpenAI) was used to improve the clarity and structure of sections of this manuscript, including suggestions for rewording sentences and rephrasing for better flow. All scientific content and interpretations are the sole responsibility of the authors.

No generative AI tool was used to fabricate data, generate references, or write substantive parts of the scientific content without human oversight and verification.

Βιβλιογραφία

Ξενόγλωσση Βιβλιογραφία

[1] [n. d.]. W3C Recommendation, SPARQL Query Language for RDF. https://www.w3.org/TR/rdf-sparql-query/. Accessed: 2025-05-10.

[2] Merve Atasever, Mihir Nitin Kulkarni, Qingpei Li, Matthew Hong, and Jyotirmoy V. Deshmukh. 2025. Multi-Agent Path Finding via Decision Transformer and LLM Collaboration. https://openreview.net/forum?id=Mvn48u0ehO

[3] Ben Athiwaratkun, Andrew Gordon Wilson, and Anima Anandkumar. 2018. Probabilistic FastText for Multi-Sense Word Embeddings. CoRR abs/1806.02901 (2018). arXiv:1806.02901 http://arxiv.org/abs/1806.02901

[4] Jiang Bao, Nan Duan, Zhi Yan, Ming Zhou, and Tiejun Zhao. 2016. Constraint-Based Question Answering with Knowledge Graphs. Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI-16) (2016), 948–954.

[5] Vicente Calisto, Benjamín Farias, Wim Martens, Carlos Rojas, and Domagoj Vrgoc. 2024. PathFinder Demo: Returning Paths in Graph Queries. In Proceedings of the ISWC 2024 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 23nd International Semantic Web Conference (ISWC 2024), Hanover, Maryland, USA, November 11-15, 2024 (CEUR Workshop Proceedings, Vol. 3828), Lorena Etcheverry,

Vanessa López Garcia, Francesco Osborne, and Romana Pernisch (Eds.). CEUR-WS.org. https://ceur-ws.org/Vol-3828/paper34.pdf

[6] Jianjun Chen, Cheng Hou, and Xi Wang. 2021. LLM-Guided Pathfinding in Knowledge Graphs: An AI-Driven Exploration Approach. In Neural Information Processing Systems (NeurIPS-21).

[7] Xiaojun Cheng, Xin Zheng, and Donghui Zhang. 2018. Dynamic SPARQL Query Expansion using Reinforcement Learning. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM-18). 765–774. doi:10.1145/3269206.3271735

[8] KENNETH WARD CHURCH. 2017. Word2Vec. Natural Language Engineering 23, 1 (2017), 155–162. doi:10.1017/S1351324916000334

[9] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. 2014. Introducing Wikidata to the Linked Data Web. In The Semantic Web – ISWC 2014, Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble (Eds.). Springer International Publishing, Cham, 50–65.

[10] Benjamín Farias, Wim Martens, Carlos Rojas, and Domagoj Vrgoc. 2024. PathFinder: Returning Paths in Graph Queries. In The Semantic Web - ISWC 2024 - 23rd International Semantic Web Conference, Baltimore, MD, USA, November 11-15, 2024, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 15232), Gianluca Demartini, Katja Hose, Maribel

Acosta, Matteo Palmonari, Gong Cheng, Hala Skaf-Molli, Nicolas Ferranti, Daniel Hernández, and Aidan Hogan (Eds.). Springer, 135–154. doi:10.1007/978-3-031-77850-6_8

[11] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeffrey F. Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: Toward Building Entity Matching Management Systems. Proc. VLDB Endow. 9, 12 (2016), 1197–1208. doi:10.14778/2994509.2994535

[12] Thomas Konrath, Thomas Gottron, and Steffen Staab. 2012. Adaptive SPARQL Query Expansion for Efficient Knowledge Graph Search. Journal of Web Semantics 14, 4 (2012), 120–136. doi:10.1016/j.websem.2012.08.002

[13] Ni Lao, Tom Mitchell, and William Cohen. 2011. Random Walk Inference and Learning in A Large Scale Knowledge Graph. In Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP-11). 529–539.

[14] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. 2015. DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web 6, 2 (2015), 167–195. doi:10.3233/SW-140134 arXiv:https://journals.sagepub.com/doi/pdf/10.3233/SW-140134

[15] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA, Blai Bonet and Sven Koenig (Eds.). AAAI Press, 2181–2187. doi:10.1609/AAAI.V29I1.9491

[16] Jan Portisch and Heiko Paulheim. 2024. The RDF2vec family of knowledge graph embedding methods. Semantic Web 15, 3 (2024), 845–876. doi:10.3233/SW-233514

[17] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. CoRR abs/1908.10084 (2019). arXiv:1908.10084 http://arxiv.org/abs/1908.10084

[18] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In Proceedings of the 16th International Conference on World Wide Web (Banff, Alberta, Canada) (WWW '07). Association for Computing Machinery, New York, NY, USA, 697–706. doi:10.1145/1242572.1242667

[19] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. Commun. ACM 57, 10 (2014), 78–85.

[20] Jingxuan Wang, Haonan Zhu, and Yuhan Li. 2022. Prompting Large Language Models for Knowledge Graph Reasoning. In Proceedings of the

36th AAAI Conference on Artificial Intelligence (AAAI-22). 8903–8910.

[21] Yuxiang Wang, Arijit Khan, Tianxing Wu, Jiahui Jin, and Haijiang Yan. 2020. Semantic Guided and Response Times Bounded Top-k Similarity Search over Knowledge Graphs. In 36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020. IEEE, 445–456. doi:10.1109/ICDE48307.2020.00045

[22] Ikuya Yamada, Akari Asai, Jin Sakuma, Hiroyuki Shindo, Hideaki Takeda, Yoshiyasu Takefuji, and Yuji Matsumoto. 2020. Wikipedia2Vec: An Efficient Toolkit for Learning and Visualizing the Embeddings of Words and Entities from Wikipedia. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. Association for Computational Linguistics, 23–30.

[23] Bishan Yang, Wen tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2017. Learning Multi-Relational Semantics Using Neural Networks. In Proceedings of the International Conference on Learning Representations (ICLR-17).

[24] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018, Yike Guo and Faisal Farooq (Eds.). ACM, 974–983. doi:10.1145/3219819.3219890