Knowledge Graph Injection for Reinforcement Learning

Robert Wardenga[1,*,†], Liubov Kovriguina[2,3,*], Dmitrii Pliukhin[4], Daniil Radyush[4], Ivan Smoliakov[4], Yuan Xue[5], Henrik Müller[5], Aleksei Pismerov[4], Dmitry Mouromtsev[6] and Daniel Kudenko[5]

[1] *Institut für Angewandte Informatik (InfAI) e. V., Goerdelerring 9 04109; Leipzig, Germany*
[2] *metaphacts GmbH, Daimlerstraße 36, 69190, Walldorf, Germany*
[3] *Fraunhofer IAIS Dresden, Schloss Birlinghoven 1, 53757, Sankt Augustin, Germany*
[4] *ITMO University, Kronverksky Pr. 49, bldg. A, St. Petersburg, 197101, Russia*
[5] *Forschungszentrum L3S, Appelstraße 9a 30167 Hannover, Germany*
[6] *TIB – Leibniz-Informationszentrum Technik und Naturwissenschaften und Universitätsbibliothek, Welfengarten 1B, 30167 Hannover, Germany*

Abstract

In reinforcement learning (RL) an agent usually learns the specifics and rules of the environment via interaction. This limits the agent in taking the best action only from the current observation and past experience. Therefore, providing relevant external knowledge for RL agents, as well as incorporating learned knowledge in the RL process can be a critical part of agent's successful training in real-world tasks. We propose a method, an architecture and experimental results for injecting expert knowledge in the form of RDF knowledge graphs (KGs) into the RL processes, showing how knowledge consumption increases sample efficiency. Furthermore, we investigate the scalability of our approach concerning the complexity of the underlying task showing injection of KGs is beneficial to the solution of more complex RL tasks. For experimental evaluation we used the Minigrid environment, which is a standard benchmark for RL. For this environment, we designed an ontology and generated a KG, that promotes reusability and interoperability across heterogeneous data of the environment. We show that adding knowledge to the agent's learning process improves sample efficiency and the benefits increase with the complexity of the environment.

Keywords

Reinforcement Learning, Knowledge Graphs, Knowledge Injection, State Representation.

1. Introduction

Recently, RL has become increasingly popular as an approach for solving a variety of problems in numerous fields. These problems include, for example, training foundation models [1], dialog management, healthcare, personalized suggestions and recommendations, gaming, self-driving cars, and many other applications in different domains. This success is a result of the progress of

Workshop at ISWC 2023 on DEEP LEARNING FOR KNOWLEDGE GRAPHS

*Corresponding author.

† R. W. appreciates funding from Federal Ministry of Education and Research (BMBF) under iDOKS + FKZ 16SV8765

wardenga@infai.org (R. Wardenga); lk@metaphacts.com (L. Kovriguina); zeionara@gmail.com (D. Pliukhin); daniil.radyush@gmail.com (D. Radyush); smol.ivan97@gmail.com (I. Smoliakov); xue@l3s.de (Y. Xue); hmueller@l3s.de (H. Müller); alekceu444@gmail.com (A. Pismerov); d.muromtsev@gmail.com (D. Mouromtsev); kudenko@l3s.de (D. Kudenko)

such methods as DQN [2], Alpha-GO [3], AlphaZero [4] and others. These methods efficiently generate abstractions and learn policies in the tasks with large state spaces.

At the same time, some tasks remain challenging for RL, when an agent does not have the opportunity to obtain in advance all the necessary knowledge about the specifics of the environment with which he has to interact and must adapt to changing conditions and somehow decide which of the knowledge, relevant to the environment, to use. Moreover, the provision, integration, and use of external knowledge for agents is also a challenge for knowledge engineering, when encoding of observations about the environment can change dynamically. Sometimes, the domain (environment) is too complex to be efficiently represented by standard RL approaches. Knowledge graphs serve as a rich representation model for knowledge, that can be expressed in terms of entities and relations, as well as larger frame structures, i.e. events. However, for complex or unseen tasks there is often no clear idea, how the knowledge should be consumed or what are the most relevant knowledge snippets for a downstream task (for example, when long term reasoning is required). The area of RL, on the contrary, brings the opposite: approaches to model behavior and decision-making process towards solving tasks via interaction with the environment. The interactive nature of RL allows to self-learn with provided knowledge and integrate various data sources without tedious data annotation and curation processes and get better results on complex knowledge-seeking tasks. The motivation of the current paper is to provide theoretical and practical ground to large scale bridging of recent advances in Semantic Web technologies and knowledge graph representations with RL pipelines.

This research setup looks a priori promising, because knowledge injection in deep neural networks has been shown to be a powerful paradigm to add external knowledge to neural architectures, allowing to ease the learning process and enabling them to reason better in downstream tasks[5],[6],[7]. However, knowledge in such scenarios has been understood in a very broad, if not say, vague, sense: expert notes, tables, data snippets, retrieved from databases, etc. In the area of language models, knowledge graph injection turned out to be especially beneficial, although the questions, which knowledge should be injected, and when, as well as the problem of heterogeneous vector spaces, are still not fully solved. We are tackling the problem of knowledge graph injection in RL pipelines trying to mitigate the challenges both in knowledge graphs and RL and opening a broad path for bridging RL and Semantic Web, where agents can improve their learning process from the existing knowledge graphs, and the latter can be refined from the interaction process of agent and environment (similar as in RLHF[8]).

Learning of RL agents derives from interaction with the environment, in which basic RL algorithms, such as Q-learning, generally require visiting a large number of available states many times before a satisfactory policy, or map from states to actions, can be achieved. Learning problems besides the "curse of dimensionality"[9] are also induced by the incompleteness and variability of knowledge about the environment in real-world applications. This limitations of the state space representations in RL can be tackled with KGs, which provide rich and expressive ways to represent and analyze weakly structured domain data and exploring domain structure.

In this article, we propose a new approach to augment observations with relevant knowledge. The knowledge is provided as domain KG embeddings, contextualized with the agent's state. This approach works independently of the underlying specific deep RL method. We evaluate our approach by extending the vanilla DQN algorithm with knowledge injection component (Sec. 4) and show performance improvements across 7 Minigrid environments [10] of different complexity (see Sec. 5.2).

The key contributions of the paper can be summarized as follows:

- **KGRL**, a method and an architecture for on-the-fly retrieving and injecting relevant knowledge from RDF KGs to augment RL pipelines.
- Open source Minigrid ontology and an associated KG generator, that may serve as a testing ground for KG injection[1].
- Experimental results, demonstrating that knowledge injection reduces the cost of learning new policies, especially for more complex environments.

The paper is structured as follows: the introduction is followed by preliminaries in Section 2 before discussing related work in Section 3, continued with a detailed description of the KGRL approach in Section 4 and experimental setup and results in Section 5, finishing with conclusions and future work in Section 6. The Appendix to the paper contains descriptions of the Minigrid environments with its complexity classes, and learning cureves, and Appendix in the repository provides information on the experimental setup, hyperparameter tuning, training runs, KGRL adaptation to other domains, Minigrid ontology visualization, and a list of abbreviations.

2. Preliminaries

In our work we consider scenarios that extend the standard RL architecture with the components required to support the agent with relevant knowledge, retrieved from knowledge graphs contextualized with the current observation. The preliminaries respectively cover the basic concepts of the approaches and methods, that were reused or extended in KGRL, highlighting how they can benefit from each other.

2.1. Reinforcement Learning and Deep Q-Learning

Reinforcement learning involves an agent interacting with an environment and making sequence of action decisions. An RL task can be modeled as a Markov Decision Process (MDP) $M = (S, A, T, R, \gamma)$, a tuple consisting of a state space S, an action space A, the transition dynamics $T(s, a, s') : S \times A \times S \to [0, 1]$, the reward function $R(s, a, s') \in \mathbb{R}$, and a discounting factor $0 < \gamma \leq 1$.

The goal of an RL agent is to learn the optimal policy which maximizes the expected discounted reward by interacting with the environment. Given an arbitrary policy π, action-value function $Q_\pi(s, a)$ indicates the expected discounted return of executing action a in the current state s and following the policy π afterwards. On the other hand, given an action-value function $Q(s, a)$, a greedy policy can be induced via selecting the action with the highest Q-value.

Q-learning [11], a classic model-free RL algorithm, finds the optimal policy π^* by learning the optimal Q-function

$$Q^*(s, a) = \max_\pi \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \mid s_t = s, a_t = a, \pi]$$

[1]We provide a repository containing additional results ablations and training runs, the ontologies for the Minigrid environment, code to generate KGs from Minigrid environments, code and a database of hyperparameters to perform all experiments as well as detailed descriptions on reproduction - https://github.com/wardengainfai/KGRL

The Q-function is iteratively updated according to Bellman Optimality equation:

$$Q_{t+1}(s_t, a_t) = (1-\alpha)Q_t(s_t, a_t) + \alpha \left[R\left(s_t, a, s_{t+1}\right) + \gamma \max_{a'} Q_t\left(s_{t+1}, a'\right) \right]$$

Deep Q-learning (DQN) [2] approximates the Q-function with a deep neural network that is characterized by its weights w. The objective function of DQN to minimize is:

$$\Delta_w = \mathbb{E}_{(s,a,s',r)\sim D}[r + \gamma \max_{a' \in A} Q_{\hat{w}}(s', a') - Q_w(s, a)]$$

where D is the experience replay buffer, which caches recent transition history and helps improve sample efficiency of learning, i.e. the agent can achieve a higher reward with the same amount of interactions or samples from the environment. $Q_{\hat{w}}$ is the target Q-network, which copies the weights of Q_w by each fixed interval of time steps. This technique has been proven to be critical for the stability of training performance.

2.2. Ontologies, Knowledge Graphs and Knowledge Graph Embeddings

Ontologies are explicit formal specifications of the terms in the domain and relations among them[12], which provide humans and machines an accurately understandable context or meaning and ensure a common understanding of information[2]. The ontology modeling languages (i.e. OWL, an ontology language for the Web, based on description logic[13]) are characterized by formal semantics, and include description of concepts in a domain of discourse (**classes** (sometimes called concepts)), properties of each concept describing various features and attributes of the concept (slots, sometimes called roles or **properties**), and **restrictions** on slots[14]. OWL has rich and expressive syntax, allowing to describe quite complex abstractions, i.e. events and temporal relations.

Knowledge graphs represent structured collections of facts describing the world in the form of relationships between entities[15]. KGs is a popular model for representing static, graph-structured data, but they do not offer an effective means to learn and discover processes, like RL does. Formally, a KG is a graph-based representation model, which nodes represent entities and edges represent relations between these entities[15], shaped as a set of triples: "for a given set of entities $\mathcal{E}$ and set of relations $\mathcal{R}$, we consider a knowledge graph $\mathcal{K} \subseteq \mathbb{K} = \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ as a directed, multi-relational graph that comprises triples $(h, r, t) \in \mathcal{K}$ in which $h, t \in \mathcal{E}$ represent a triples' respective head and tail entities and $r \in \mathcal{R}$ represents its relationship."[16].

Among the chief benefits of OWL/RDF KGs is their formal representation richness, reasoning capabilities, large inventory of validation and visualization tools, reusability, and inherent capability for flexible evolving and lifelong population. Representation richness, established methods for incompleteness handling and never ending population of knowledge graphs can be considered as most beneficial for the RL applications. Moreover, a well-elaborated pool of KG embedding algorithms has opened frontiers for its injection into (large) language models and other NLP and ML pipelines[17].

Knowledge Graph Embedding Models (KGEMs) learn latent representations for entities $e \in \mathcal{E}$ and relations $r \in \mathcal{R}$ in a KG that best preserve its structural properties[16],[18] within an Embedding space $\mathbb{X}$ (often $\mathbb{X} = \mathbb{R}^{n_{emb}}$ with embedding dimension n_{emb}). Authors in [16]

[2]https://cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/owl-101/

define KGEM as four components: an *interaction model*, a *training approach*, a *loss function*, and its usage of *explicit inverse relations*. An interaction model $f : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \to \mathbb{R}$ computes a real-valued score representing the plausibility of a triple $(h, r, t) \in \mathbb{K}$ given the embeddings for the entities and relations. In general, a larger score indicates a higher plausibility of a triple, but, being model-dependent, it cannot be directly interpreted as a probability. The pool of KGEMs is quite diverse: translation models, tensor factorization models, semantic matching models, hyperbolic embeddings[19].

In KGRL, we start with the most general and adopted translation models (TransE, TransR, RotateE, etc.), which show good performance across tasks. TransE[20] is an energy-based model for learning low-dimensional embeddings of entities, that models relations as translations of head to tail embeddings, i.e. $h + r \approx t$. Following the notation in [16], the interaction model is defined as: $f(h, r, t) = -\|h + r - t\|_p$, with $p \in \{1, 2\}$ is a hyper-parameter. Despite its simplicity, TransE is known for computational efficiency and good performance for modeling most kinds of relationships. However, it inherently cannot model 1-N, N-1, and N-M relations.

Linking Text and Knowledge Graphs Entity and relation linking (ER), which task lies in grounding text fragments to entities and relations of a specific KG[21],[22], serves as a bridge between unstructured text and KGs. Constant evolving of KGs and language require entity and relation linking algorithms to be flexible and adaptive, able to handle out-of-vocabulary and rare cases. ER linking is a crucial part of most approaches to knowledge injection in language models (LMs). Knowledge injection aims to augment LMs with missing knowledge, improving LMs awareness about the missing facts and mitigating its incompleteness. Multiple interesting approaches have appeared for this task, employing joint training, vector space alignment, alignment on text level, and other methods[6],[23],[24]. This direction in combination with RL opens new frontiers in training more knowledge-aware models, handling heterogeneous data.

In multimodal KGs (i.e. combining text, image, geo-spatial data [25]) one of the modalities can often serve as the interlinking layer for the others (i.e., speech and image can be interlinked via text modality). This has been explored deeply for entity and relation extraction and linking from text but also for linking images with KGs [26]. Advances in multimodal ER linking can be especially helpful to ground natural language tasks in heterogenous RL environments, allowing linking of multimodal environment observations to KG.

3. Related Work

Provision of knowledge to an RL agent has been studied from different perspectives. Some approaches concentrate on keeping the knowledge of the past experience (i.e., from interaction with the textual environment in [27]) or environment topology in the agent's memory. For instance, Humphreys et al. [28] propose an approach in which agents can utilise large-scale context sensitive database lookups to support their parametric computations, concluding that large scale retrieval leads to performance boost.

Xue et al.[29] treat the Markov Decision Process (MDP) as a graph and employ graph learning methods to encode knowledge about its topology into an embedding space, upon which an abstraction of the state space is generated via clustering methods. An abstract MDP (AMDP) is then constructed, which can be solved exactly using dynamic programming. The resulting value function is used as a source of reward shaping for solving the original MDP, which helps to improve the sample efficiency of the RL agent. This method provide a pipeline to automatically generate an AMDP to help speed up RL. However, constructing and solving the AMDP needs to be done before the real RL policy optimization starts and that introduce additional overhead to RL learning. dynamic programming does not scale up well. Burden and Kudenko [30] also seek to construct an abstract MDP to induce reward shaping. The abstraction is generated by uniformly partitioning the state space. The cost of building AMDP is reduced but topology of the state space is ignored, which can lead to misleading reward shaping.

In KG-DQN [27] the authors concern themselves with solving text-based games, compounding local perceptual knowledge in a KG. The method proposed in [31] utilises a commonsense KG, that exhibits certain similarity to our approach. While [31] also adds commonsense knowledge via linking, [32], [27] and [31] approaches use GNNs to feed the obtained knowledge to a specific RL-model. In contrast, our approach uses KGEs and provides knowledge in a RL-model agnostic way, that allows to integrate knowledge into a wide range of RL methods easily. The closest approach to KGRL was proposed for recommendation systems in [32], where authors use prior knowledge to enrich the representation of items and user states and guide the candidate selection for better candidate item retrieval. However, the proposed KGRL architecture has a number of benefits in comparison to that paper. KGRL allows broader interaction between modalities (image, text, etc.) with less training and is easier for configuring, being implemented in "lego" style: formats and technologies choice allow to joggle with different RL methods, LLMs, KGEMs and KGs). Furthermore, we investigate the scaling w.r.t. the complexity of the underlying task.

4. Approach

4.1. Overview

The proposed KGRL approach aims to augment observation vectors with KG embeddings, retrieved from the domain KGEM and external LLMs based on the current agent's state, that are injected during the learning process. This approach works independently of the underlying deep RL method and is applicable to any RL method, where state and observation can be represented as vectors.

The approach performs step-wise knowledge injection to the RL model at the observation level using a **KG wrapper**. Within the KG wrapper there are two main types of components: **linkers**, which ground non-structured data in the observation to the KG snippets (subgraph), and **retrievers**, which retrieve relevant embeddings for the input subgraph. The process goes as follows: at each time step the observation is split by data type, which it contains, and these parts of observation are augmented with knowledge snippets by linkers:

- *text* data are processed by entity and relation linker, which returns a list of relevant triples of the KG,
- *image* data, i.e. describing the view of the agent, are processed by object detection linker, which also returns a list of entities,
- *additional state representations*, that can contain heterogeneous data, and may require specific environment-dependent linkers, i.e. geo-spatial linker, bio-chemical linker.

These extracted knowledge snippets are routed to the knowledge retriever, which retrieves KGEs, relevant to these snippets (Sec. 4.2.2). Thus, at each iteration the agent observes only the knowledge relevant to its state. The KGRL architecture is shown in Figure 1 for the Deep Q-Network.

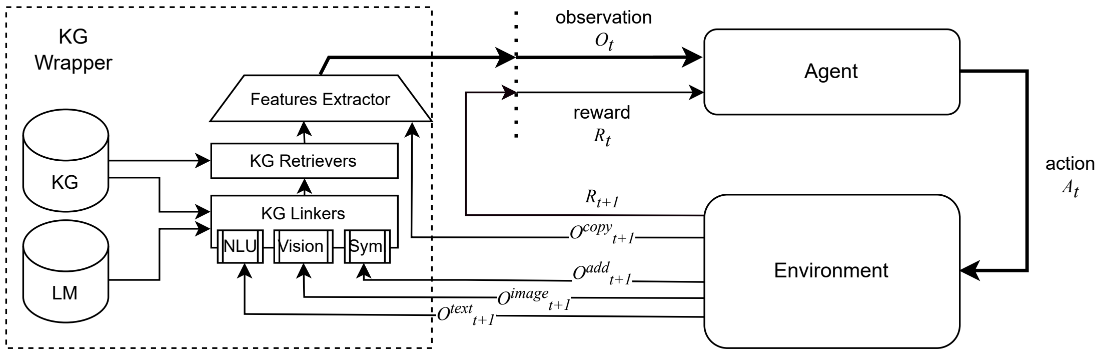


Figure 1: Knowledge Injection for Reinforcement Learning (KGRL): Method Overview. The image describes the knowledge injection process for a general RL algorithm, that operates with states, observations and rewards. O^{text} - text data, O^{add} - heterogeneous data, O^{graph} - KGEs and RDF triples, O^{image} - image data.

On the example of Q-learning, we formally describe the adaptions made in KGRL. The Q-function update is only modified with an additional input f_{KG}

$$\begin{aligned} Q_w^{t+1}(s, a, f_{KG}) &:= Q_w^t(s, a, f_{KG}) \\ &+ \alpha \left(r + \gamma \max_{a' \in A} Q_w^t(s', a', f'_{KG}) - Q_w^t(s', a, f'_{KG}) \right), \end{aligned}$$

where s is the current state and $f_{KG} := \mathbf{F}_{KG}(s)$ are the KG based features extracted from state s. We can further decompose $\mathbf{F}_{KG}\colon\ : S \to \mathbb{X}^{n_r}$ into:

$$\mathbf{F}_{KG}(s) = \mathbf{R}_* \left(\mathbf{E}_*(s) \right),$$

with $\mathbf{E}_*\colon S \to \mathcal{P}(\mathcal{E} \times \mathcal{R})$ denoting an extractor, mapping the state space S into the powerset of the productspace of entities $\mathcal{E}$ and relations $\mathcal{R}$ in the KG and $\mathbf{R}_*\colon \mathcal{P}(\mathcal{E} \times \mathcal{R}) \to \mathbb{X}^{n_r}$ a retriever, mapping to the product embedding space.

4.2. KGRL Components

We implement our approach as an observation wrapper, called KG wrapper, over the original environment. The main components of this wrapper are (1) resources, such as the KG and LLMs, (2) KG linking, (3) KG retrieval, (4) KG embedding of the retrieved subgraph.

Figure 1 provides an overview of our approach and the components included in the KG wrapper. In the following subsections we describe the KG linking and KG retrieval in more detail. The process for adapting KGRL to a new domain is described in the Appendix C.

4.2.1. Knowledge Graph Linking

Entity and Relation Linking in KGRL For this purpose, KGRL uses one of pre-trained SBERT models [3] tailored to Semantic Textual Similarity task, which facilitates entities and relations retrieval from a knowledge graph for a specified environment. Specifically, each triple in the knowledge graph is represented by concatenation of its subject and object labels, and the model evaluates cosine similarity between triples labels and n-grams from the textual description of environment (e.g. mission string for Minigrid). Subsequently, a given number of the most relevant knowledge graph triples provide additional information about the environment in the form of its embeddings and the confidence scores (cosine similarity metrics) for the agent to enhance its decision making process (an illustration of this process can be found in the repository).

Symbolic KG linking Heterogeneous and environment specific state representations (i.e., codes, coordinates) can be linked to a KG via symbolic linking, which represents a character based matching, agnostic of the underlying semantics. In case of the Minigrid environment additional state representations are given as arrays corresponding to cells in the grid. We use this representation to link a cell in the grid to a cell entity in the KG.

4.2.2. Knowledge Graph Retrieval

In order to make the information in the KG available to the agent we first have to retrieve a subset of the graph relevant to the current state of the agent. This happens on the basis of the extractions from the original observation. The extractions are fed into the retrieval methods which return relevant subsets of the KG as embedding vectors. This additional information allows the agent to navigate in the latent high dimensional Knowledge Graph embedding space in order to complete tasks. This is a crucial part in the pipeline as the underlying knowledge base can be huge and passing the entire KG might be unfeasible. Below we introduce two methods for retrieval:

k-NN Retrieval The k-nearest neighbors of the extracted entities in the KGE space are determined including the extracted entities. This method returns embeddings of semantically similar entities, as well as entities that are connected to the extracted entities by a short path. As the computation of a k-NN index can be done in advance, this method scales well in the number of steps.

Random walk Retrieval This method retrieves random walks of fixed length, starting in the extracted entities, consisting of embeddings of encountered entities and relations. This is computationally slightly more expensive than the k-NN look-up, but has the advantage allowing a deeper exploration of the latent KGE space due to the inherent stochasticity of this process.

5. Experiments

5.1. Experimental Setup

To evaluate our hypotheses, we perform a wide range of experiments on environments of different complexities. We run the training cycles for the baseline as well as for different configurations of our proposed method. For each environment, we conduct a hyperparameter search

[3]https://www.sbert.net/

prior to training in order to find decent training parameters for the baseline algorithm. This ensures a possibility for convergence for the baseline. We do not perform further hyperparameter search for our proposed modifications. Further improvements are likely possible by searching for the best knowledge graph embedding model for a given domain, as well as tuning the dimension and the size of the retrieved subset of the KG.

5.2. Running KGRL on the MiniGrid Environment

MiniGrid Ontology For the Minigrid environment we design an OWL ontology, which completely describes Minigrid environments from the perspective of available tasks. We describe artifacts, states and properties (i.e. *colour* and *location*) and agent's actions and the relations between them. The visualized ontology and OWL/TTL files can be found in the repository and Appendix D.1.

For example, the ontology describes, which doors (class *Door*) the room (class *Room*) has, in which cell (class *Cell*) it is located, what its state is (subclasses *Opened*, *Locked* or *Closed* of class *state*) and what color it is assigned to. On the other hand, a color can also be assigned to keys (class *Key*) creating a short connection between doors and keys they can be opened with.

Knowledge Graph Generation from Environment Representation Based on the above ontology we automatically generate a KG from the environment representation. The representation is given as an array of size *grid size* $\times$ 3 giving for each cell its content, the state of the content and its color. The generated KG describes the environment completely in terms of positions, states and interactions of objects. We use this KG as an exterior knowledge that the agent can explore in parallel to the environment.

Extraction and Retrieval For retrieving from the KG we associate the grid cell, where the agent is currently located, with the corresponding cell entity in the KG and retrieve a relevant subset of the entire KG with the methods described in 4.2.2 starting from this entity. Furthermore, as this environment returns a static mission string (see table A), we also augment our approach with a transformed mission, which consists of the most relevant triples $(h_i, r_i, t_i))_{i=0}^{i=n_triples}$ in the KG, along with their confidence scores $(c_i)_{i=0}^{i=n_triples}$, where $h_i, t_i \in \mathcal{E}$ are entities found in the mission string. We use the translational embedding model TransE to embed the retrieved subset as this model showed good performance over different applications [16].

5.3. Training Runs and Evaluation

We select environments featuring tasks of varying complexity and size to perform our experiments. In each of these environments we perform a training run for ten different seeds to verify our approach on different samples of the environments. During training, we evaluate the trained policy 200 times at regular intervals. The average episode length (i.e. the number of steps to complete a task) as well as the average rewards achieved in the evaluation episodes are reported. We additionally compute the average number of training timesteps needed to achieve a given reward. By averaging over different Environments and concentrating on relative timesteps based on the pointwise best achieved training steps, this allows to draw conclusions about the average gain in terms of sample efficiency. This procedure is performed for the Baseline - for which we choose DQN (in the following referred to as `plain`) - and two configurations of our approach. The first approach `knn` consists of the inclusion of k-nearest neighbors to the current state of the agent. The second approach `rw` augments the observation by including the knowledge graph embeddings of entities and relations encountered in a random walk of length k starting from the current state and the third approach `rw-ner` additionally returns embeddings of the triples extracted from the mission string in Minigrid.

5.4. Results

The results of our training runs are reported in Figure 3 in the Appendix, along with further training runs evaluated with perturbed actions. Combining the results across environments in one complexity class we find that the performance gain increases with the size of the underlying grid and the complexity of the Task, see Figure 2. This is further highlighted in Figure 3 in the Appendix, which shows that the average number of training steps to reach the highest reward is half of that of the baseline without Knowledge Graph injection. For the most complex environments knowledge injection leads to the highest rewards in a third of the training steps on average (see also Figure 2).

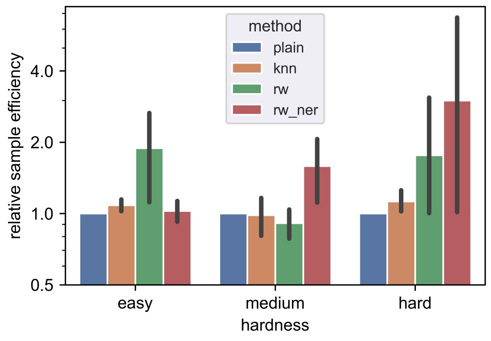


Figure 2: Relative sample efficiency of learning with respect to reward performance, $t_{plain}^{r_{plain}}/t_{*}^{r_{plain}}$, where $t^{r_{plain}}$ are the timesteps needed to achieve the best reward r_{plain} with the plain DQN and $t_{*}^{r_{plain}}$ the timesteps needed by the respective approach $*$ to achieve the same reward (higher is better). Bars represent averages over all environments in the complexity category.

6. Conclusion and Future Work

Our experimental results show, that KGRL method significantly reduces the amount of training steps needed to achieve high rewards, showing better results for more complex environments, where decision making by the agent requires more knowledge to adapt. At the same time, we see that techniques of KGs injection differ in efficiency, depending on the specificity and structure of KGs. For example, simpler environments are characterized by the presence of topological information in the KGs, while complex environments include also more domain-specific rules (e.g., using certain keys for given doors). All of this raises additional questions to explore the ways to retrieve, encode, and inject knowledge into RL pipelines. We believe that this effect confirms our hypothesis, that KG injection in RL pipelines at scale improves decision making in complex domains. In future work we will explore how the quality and completeness of the used KG influences training parameters, testing also if RL can be used to evaluate the quality of ontology creation approaches and fact extraction. Another promising research direction lies in applying KGRL to other domains, and experimenting with more elaborated approaches for integrating knowledge graphs with LLMs in a single pipeline for better alignment.

References

[1] A. A. Team, J. Bauer, K. Baumli, S. Baveja, F. Behbahani, A. Bhoopchand, N. Bradley-Schmieg, M. Chang, N. Clay, A. Collister, et al., Human-timescale adaptation in an open-ended task space, arXiv preprint arXiv:2301.07608 (2023).

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602 (2013).

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, nature 529 (2016) 484–489.

[4] H. Zhang, T. Yu, Alphazero, in: Deep Reinforcement Learning, Springer, 2020, pp. 391–415.

[5] N. Muralidhar, M. R. Islam, M. Marwah, A. Karpatne, N. Ramakrishnan, Incorporating prior domain knowledge into deep neural networks, in: 2018 IEEE international conference on big data (big data), IEEE, 2018, pp. 36–45.

[6] W. Liu, P. Zhou, Z. Zhao, Z. Wang, Q. Ju, H. Deng, P. Wang, K-bert: Enabling language representation with knowledge graph, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, 2020, pp. 2901–2908.

[7] A. Lauscher, O. Majewska, L. F. Ribeiro, I. Gurevych, N. Rozanov, G. Glavaš, Common sense or world knowledge? investigating adapter-based knowledge injection into pretrained transformers, arXiv preprint arXiv:2005.11787 (2020).

[8] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, D. Amodei, Deep reinforcement learning from human preferences, Advances in neural information processing systems 30 (2017).

[9] R. Bellman, Dynamic programming, Technical Report, RAND CORP SANTA MONICA CA, 1956.

[10] M. Chevalier-Boisvert, L. Willems, S. Pal, Minimalistic gridworld environment for gymnasium, 2018. URL: https://github.com/Farama-Foundation/Minigrid.

[11] C. J. Watkins, P. Dayan, Q-learning, Machine learning 8 (1992) 279–292.

[12] T. R. Gruber, A translation approach to portable ontology specifications, Knowledge acquisition 5 (1993) 199–220.

[13] W. O. W. Grou, Owl 2 web ontology language document overview (second edition), 2012. URL: https://www.w3.org/TR/owl2-overview/.

[14] N. Noy, D. L. McGuinness, et al., Ontology development 101, Knowledge Systems Laboratory, Stanford University 2001 (2001).

[15] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, et al., Knowledge graphs, ACM Computing Surveys (CSUR) 54 (2021) 1–37.

[16] M. Ali, M. Berrendorf, C. T. Hoyt, L. Vermue, M. Galkin, S. Sharifzadeh, A. Fischer, V. Tresp, J. Lehmann, Bringing light into the dark: A large-scale evaluation of knowledge graph embedding models under a unified framework, IEEE Transactions on Pattern Analysis and Machine Intelligence (2021).

[17] P. Colon-Hernandez, C. Havasi, J. Alonso, M. Huggins, C. Breazeal, Combining pre-trained language models and structured knowledge, 2021. arXiv:2101.12294.

[18] Q. Wang, Z. Mao, B. Wang, L. Guo, Knowledge graph embedding: A survey of approaches

and applications, IEEE Transactions on Knowledge and Data Engineering 29 (2017) 2724–2743.

[19] I. Chami, A. Wolf, D.-C. Juan, F. Sala, S. Ravi, C. Ré, Low-dimensional hyperbolic knowledge graph embeddings, arXiv preprint arXiv:2005.00545 (2020).

[20] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, Advances in neural information processing systems 26 (2013).

[21] M. Dubey, D. Banerjee, D. Chaudhuri, J. Lehmann, Earl: joint entity and relation linking for question answering over knowledge graphs, in: International Semantic Web Conference, Springer, 2018, pp. 108–126.

[22] P. Le, I. Titov, Improving entity linking by modeling latent relations between mentions, arXiv preprint arXiv:1804.10637 (2018).

[23] Z. Wang, J. Zhang, J. Feng, Z. Chen, Knowledge graph and text jointly embedding, in: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, pp. 1591–1601.

[24] B. He, D. Zhou, J. Xiao, Q. Liu, N. J. Yuan, T. Xu, et al., Integrating graph contextualized knowledge into pre-trained language models, arXiv preprint arXiv:1912.00147 (2019).

[25] Y. Liu, H. Li, A. Garcia-Duran, M. Niepert, D. Onoro-Rubio, D. S. Rosenblum, Mmkg: multi-modal knowledge graphs, in: The Semantic Web: 16th International Conference, ESWC 2019, Portorož, Slovenia, June 2–6, 2019, Proceedings 16, Springer, 2019, pp. 459–474.

[26] A. L. Tuán, T.-K. Tran, D. M. Nguyen, J. Yuan, M. Hauswirth, D. L. Phuoc, Visionkg: Towards a unified vision knowledge graph, in: O. Seneviratne, C. Pesquita, J. Sequeda, L. Etcheverry (Eds.), Proceedings of the ISWC 2021 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice (ISWC 2021), October 24-28, 2021, volume 2980 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021. URL: http://ceur-ws.org/Vol-2980/paper362.pdf.

[27] P. Ammanabrolu, M. Riedl, Playing text-adventure games with graph-based deep reinforcement learning, in: Proceedings of the 2019 Conference of the NAACL: HLT, Association for Computational Linguistics, 2019, pp. 3557–3565. doi:10.18653/v1/N19-1358.

[28] P. C. Humphreys, A. Guez, O. Tieleman, L. Sifre, T. Weber, T. Lillicrap, Large-scale retrieval for reinforcement learning, arXiv preprint arXiv:2206.05314 (2022).

[29] Y. Xue, D. Kudenko, M. Khosla, Graph learning based generation of abstractions for reinforcement learning., 2021.

[30] J. Burden, D. Kudenko, Using uniform state abstractions for reward shaping with reinforcement learning, in: Workshop on Adaptive Learning Agents (ALA) at the Federated AI Meeting, 2018.

[31] K. Murugesan, M. Atzeni, P. Shukla, M. Sachan, P. Kapanipathi, K. Talamadupula, Enhancing text-based reinforcement learning agents with commonsense knowledge, arXiv preprint arXiv:2005.00811 (2020).

[32] S. Zhou, X. Dai, H. Chen, W. Zhang, K. Ren, R. Tang, X. He, Y. Yu, Interactive recommender system via knowledge graph-enhanced reinforcement learning, in: Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval, 2020, pp. 179–188.

Appendix A Minigrid Environment

Minigrid [10], formerly known as Gym-Minigrid, is a grid world environment with a variety of configurations and tasks. A sparse reward of 1 minus a small value for each interaction step with the environment is obtained once the given task has been completed. The interaction penalty is chosen in such a way that the reward is positive upon completion. The default observation returned by the Minigrid environments consists of a the restricted view of the agent, the direction he is facing and a mission string, which does not change once the environment is initialized. In the following paragraphs we shortly describe a selection of the environments within Minigrid.

We divide the environments into three classes (*simple*, *medium* and *high*) according to its complexity, estimated via properties of the knowledge graph, which was generated for that environment, such as number of nodes, entities and relations, etc. The distribution of environments into complexity classes is shown in the Appendix A.1.

Empty and *Four Rooms* are simple environments without additional objects in the grid and with the task to navigate to a goal cell. *MultiRoom* environment also contain doors that have to be opened by the agent by the agent between a flexible number of rooms in addition *Door Key* environments also contain a key and locked doors. Here the agent first has to collect the key in order to open the door that separates him from the goal. *Lava Gap* as well as *Lava Crossing* environments require the agent to navigate to a goal while avoiding the lava, which on entering would terminate the episode with zero reward. In addition the *Key Corridor*, *Obstructed Maze* and *Blocked Unlock Pickup* environments include further objects the agent has to interact with to complete the task. For instance there are balls and boxes that can be moved to unblock a connection or boxes that contain other objects.

The minigrid Environments come with a Mission string describing the Task, see Table A

Environment	Mission String
Empty	get to the green goal square
DoorKey	use the key to open the door and then get to the goal
Lava Gap Lava Crossing	avoid the lava and get to the green goal square
Key Corridor	pick up {color} {object}
Obstructed Maze	pick up the blue ball

Table 1
Minigrid environments along with the mission strings associated with them.

A.1 Complexity of Minigrid Environments

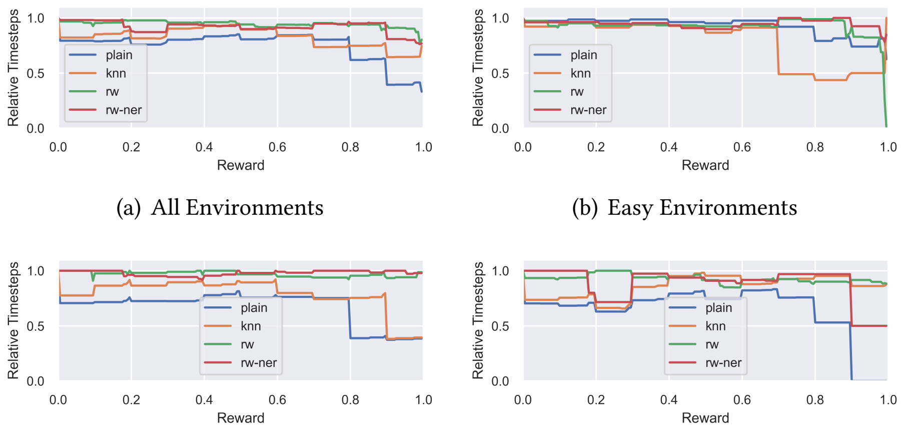
Table 2 describes the distribution of environments across 3 complexity classes (easy, medium and hard) according to the complexity of the knowledge graph, describing this environment.

Environment	N, nodes	N, edges	N, artifacts	Obstacles	Complexity
Lava Gap s5	84	140	0	yes	Easy
Lava Gap s7	156	288	0	yes	Medium
Door Key 5x5	86	132	1-2	no	Easy
Door Key 8x8	155	319	1-2	no	Medium
Lava Crossing	253	492	0	yes	Hard
Key Corridor s3r2	116	182	1-2	no	Hard
Obstructed Maze D1	775	1465	3+	yes	Hard

Table 2
Distribution of Minigrid Environments into Complexity Classes

Appendix B Learning curves

Training Runs with Random Action We aggregate the training results for different task complexities in 3.



(a) All Environments

(b) Easy Environments

(c) Medium Environments

(d) Hard Environments

Figure 3: Average of the relative number of training timesteps $(t/t^{best})^{-1}$ needed to achieve a reward, based on the least amount of training steps t^{best} to achieve that reward in the same Environments (curves close to $y = 1$ are better). The average is computed over all environments, easy environments, medium environments and hard environments.