Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning (Ammanabrolu and Riedl, 2019)

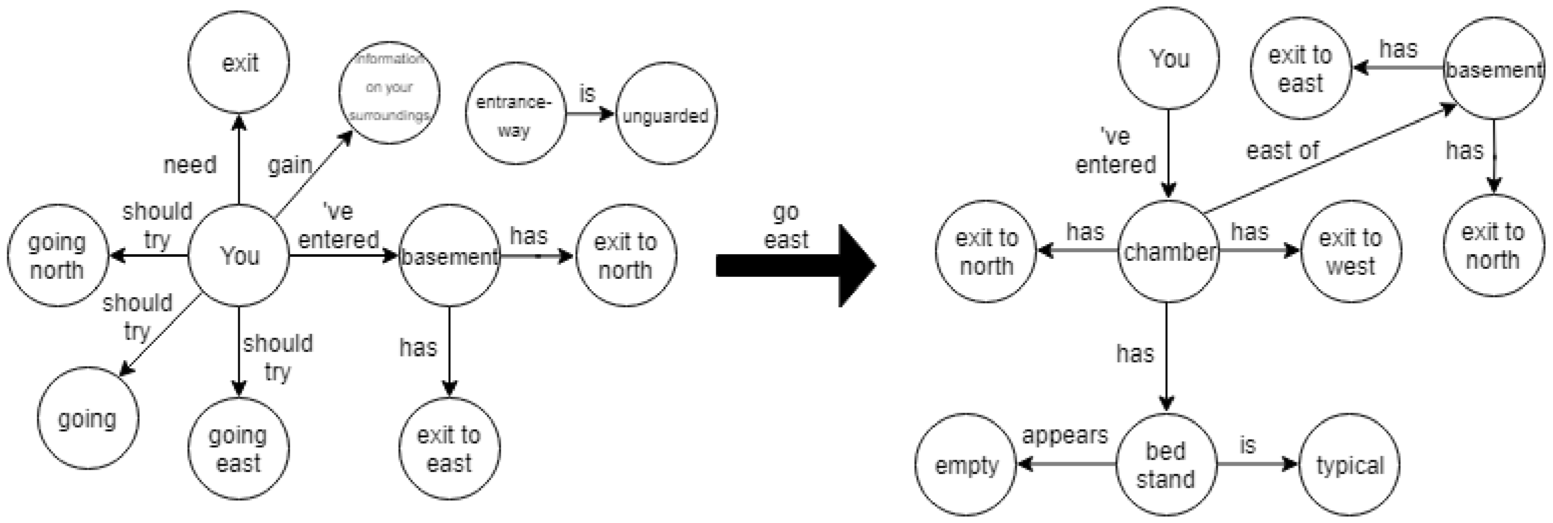
Basic Approach

- Q&A approach is adopted for the task
- State is represented in the form of a knowledge graph
- Such representation is effectively used to reduce an action space
- The knowledge graph provides a persistent memory of the world over time
- DQN method is combined with the knowledge graph (KG-DQN)

Knowledge Graph Representation

- The agent learns a set of triples, <subject, relation, object>
- These triples are extracted from the observations using Stanford’s Open Information Extraction (OpenIE)
- The knowledge graph is updated after every agent action
- A special node—designated “you”—represents the agent

KG representation example



You've entered a basement. You try to gain information on your surroundings by using a technique you call "looking." You need an unguarded exit? You should try going east. You don't like doors? Why not try going north, that entranceway is unguarded.

You've entered a chamber. You can see a bed stand. The bed stand is typical. The bed stand appears to be empty. There is an exit to the north. Don't worry, it is unblocked. There is an unblocked exit to the west.

Action pruning

- The knowledge graph is used to prune the combinatorially large space of possible actions
- Given the current state graph representation G_t, the action space is pruned by ranking the full set of actions and selecting the top-k.
- Our action scoring function is:
  - +1 for each object in the action that is present in the graph
  - +1 if there exists a valid directed path between the two objects in the graph

Node features

- Node features, *H*
  - $H = \{\mathbf{h_1}, \mathbf{h_2}, \dots, \mathbf{h_N}\}, \mathbf{h_i} \in \mathbb{R}^{\boldsymbol{F}}$
  - *N* is the number of nodes
  - *F* is the number of features in each node
- Each of the node feactures consist of the averaged word embeddings for the tokens in that node

Graph attention weights

- The attention mechanism is set up on the nodes

$$e_{ij} = LeakyReLU(\mathbf{p} \cdot W(\mathbf{h_i} \oplus \mathbf{h_j}))$$

- $\mathbf{p} \in \mathbb{R}^{2F}$ is learnable parameter and $W \in \mathbb{R}^{2F \times F}$ is a learnable linear transformation
- The attention coefficients are computed using softmax function

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k \in \mathcal{N}} exp(e_{ik})}$$

Graph attention embedding

- When graph G_t is obtained, the graph is embedded into a single vector g_t

$$\mathbf{g}_{\mathrm{t}} = f(W_g(\|_{k=1}^{K}\sigma(\sum_{j\in\mathcal{N}}\alpha_{ij}^{(k)}\mathbf{W}^{(k)}\mathbf{h}_j)) + b_g)$$

- *K* refers to the parameters of the k-th independent attention mechanism
- W_g and b_g are wegiths and biases of the component’s output linear layer
- ǁ indicates concatenation

State representation

- An encoded representation of the observation o_t is computed using a Sliding Bidirectional LSTM (SB-LSTM)
- The final state representation s_t is computed as:

$$\mathbf{s_t} = f(W_l(\mathbf{g_t} \oplus \mathbf{o_t}) + b_l)$$

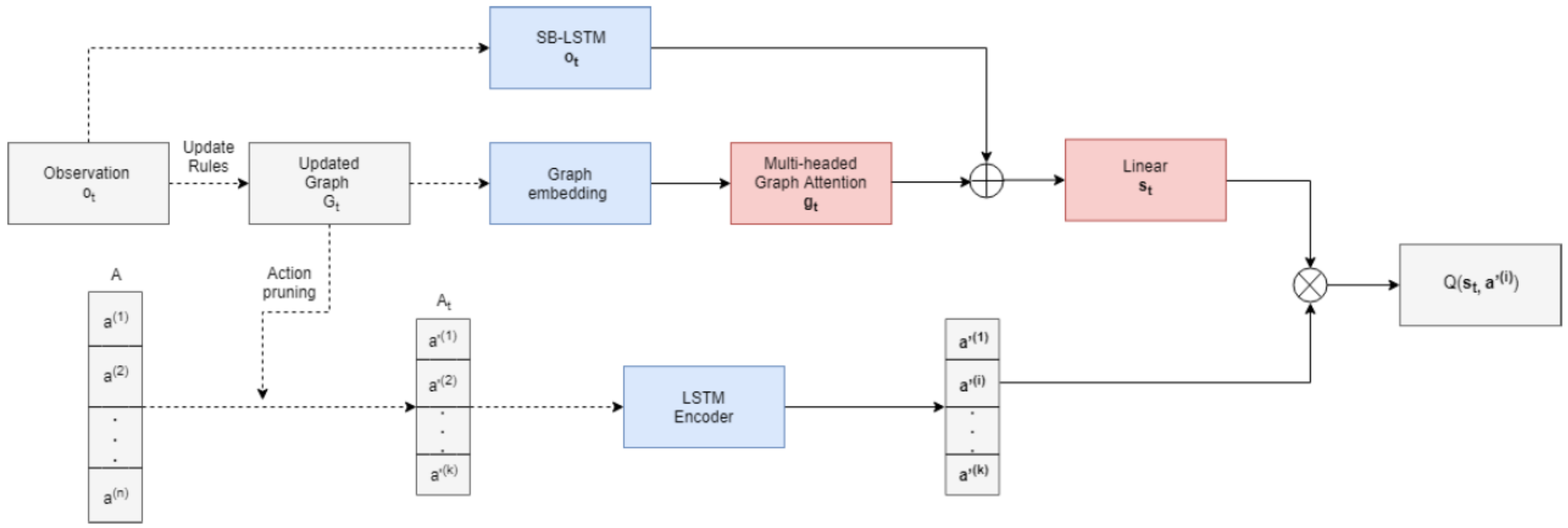
- W_l and b_l are the final linear layer's weights and biases

Action selection

- The entire set of possible actions is pruned by the action scores
- Then the action strings are embedded using an LSTM encoder
- The final *Q*-value for a state-action pair is:

$$Q(s_t, a_t) = s_t \cdot a_t$$

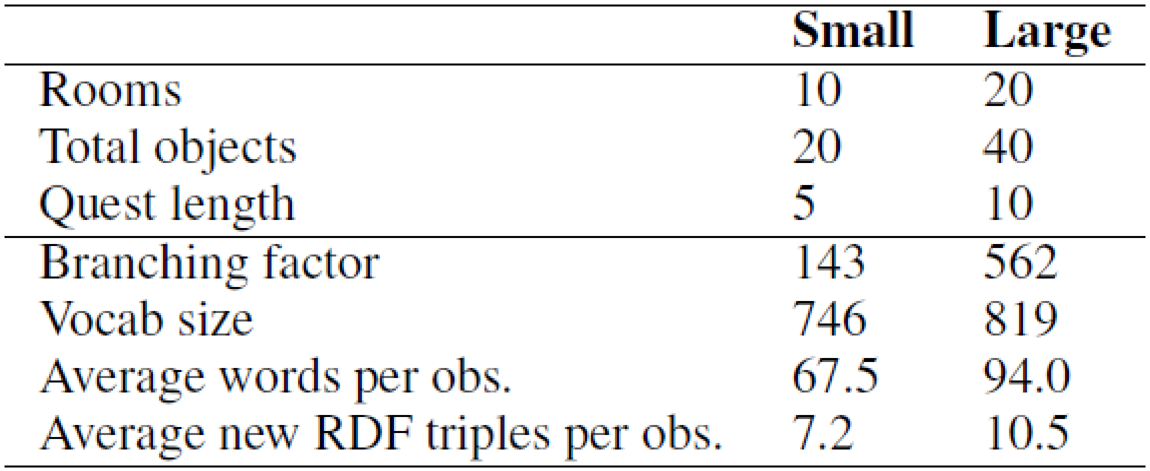
KG-DQN architecture



- Blue shading indicates components that can be pre-trained and red indicates no pre-training. The solid lines indicate gradient flow for learnable components

Experiment environment

- Games are generated with TextWorld
- Agents are not allowed to access initial instructions
- Two sets of games

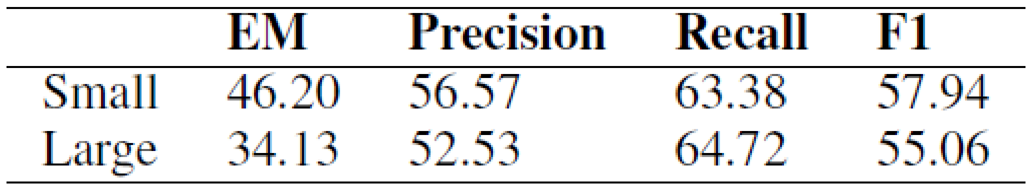


	Small	**Large**
Rooms	10	20
Total objects	20	40
Quest length	5	10
Branching factor	143	562
Vocab size	746	819
Average words per obs.	67.5	94.0
Average new RDF triples per obs.	7.2	10.5

Pre-training using Q&A library

- Usesd DrQA method (Chen et al., 2017) to train a paired question encoder and an answer encoder
- The weigths from the encoder in the DrQA system are used to initialize the weights of the SB-LSTM of the proposed system
- DrQA embedding layers are initialized with GloVe

Table 2: Pre-training accuracy.



	EM	Precision	Recall	F1
Small	46.20	56.57	63.38	57.94
Large	34.13	52.53	64.72	55.06

Experiment conditions

- Three Baseline conditions
  - Random command selection from admissible actions
  - LSTM-DQN
  - Bag-of-Words DQN
- Three versions of KG-DQN
  - Un-pruned actions with pre-training
  - Pruned actions without pre-training
  - Pruned actions with pre-training (full)

Experiment results (small)

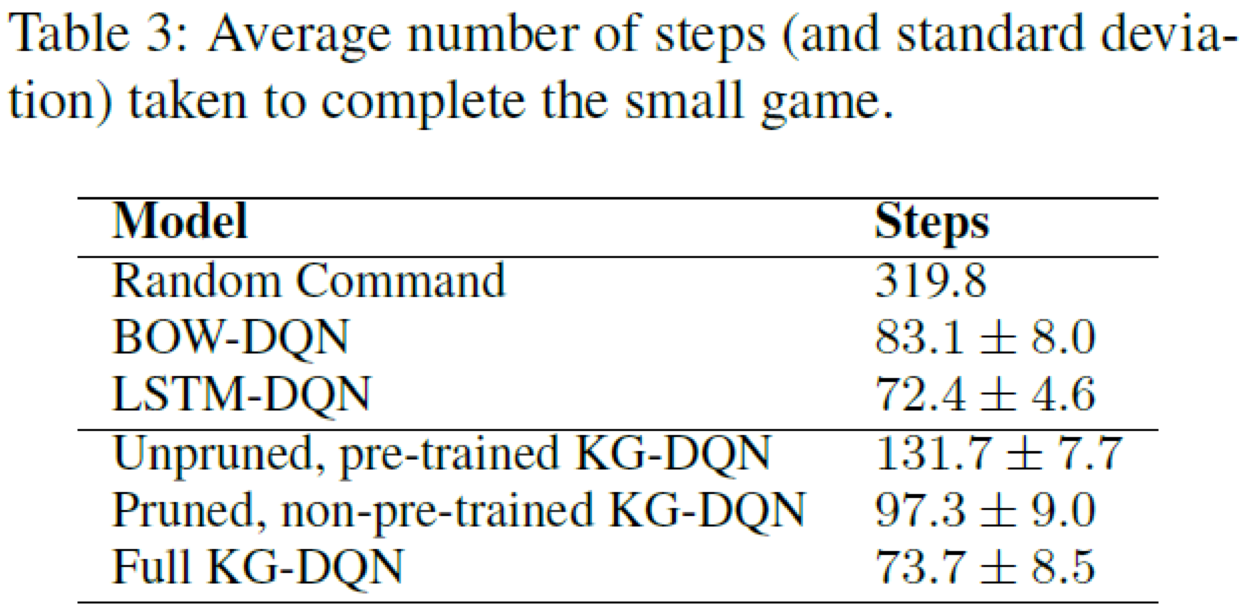


Table 3: Average number of steps (and standard deviation) taken to complete the small game.

Model	**Steps**
Random Command	319.8
BOW-DQN	83.1 ± 8.0
LSTM-DQN	72.4 ± 4.6
Unpruned, pre-trained KG-DQN	131.7 ± 7.7
Pruned, non-pre-trained KG-DQN	97.3 ± 9.0
Full KG-DQN	73.7 ± 8.5

Experiment results (large)

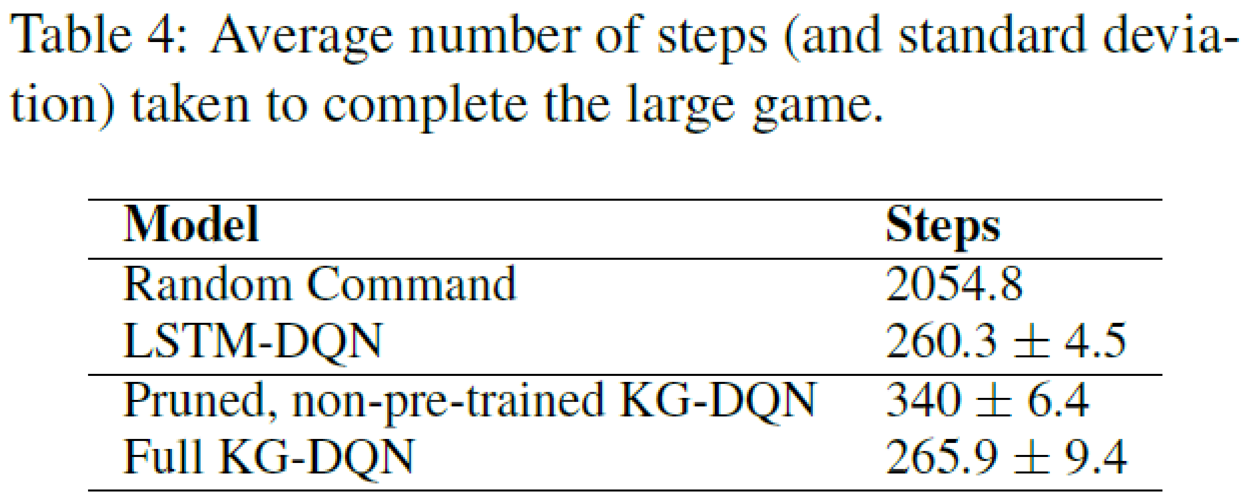


Table 4: Average number of steps (and standard deviation) taken to complete the large game.

Model	**Steps**
Random Command	2054.8
LSTM-DQN	260.3 ± 4.5
Pruned, non-pre-trained KG-DQN	340 ± 6.4
Full KG-DQN	265.9 ± 9.4

Summary of the results

- On small and large maps, all versions of KG-DQN converged faster than baselines
- KG-DQN converges 40% faster than baseline on the small game
- The KG is the main factor helping convergence time
- The ablated versions of KG-DQN require many more steps to complete quests