Reasoning over Gene Ontologies with Reinforcement Learning

Utkarsh Contractor *utkarshc@stanford.edu*, Ahmed Shuaibi *ashuaibi@stanford.edu*

I. Introduction

Our goal is to perform path link prediction for unsound queries over The Gene Ontology. Particularly we aim to predict links between entities that are not already entailed by the relations in the knowledge graph. The Gene Ontology defines genes, their products, their functions, and relationships between the gene and gene product entities. Furthermore, it provides a baseline classification of its entities into one of three categories. First, **molecular functions** inform us of molecular activities of genes and their products (for example, the function of proteins formed from a particular gene). Second, **cellular components** detail the location of genes and their products. Third, **biological processes** inform us of the larger processes comprised of the activities of multiple gene products. Although the cellular component that a specific gene or protein resides in is typically well known and categorized, this is not the case for the molecular function and biological processes for these entities. Concerning ourselves with proteins alone, there are more than 40% of proteins of unknown function, formally known as *PUFs*[5]. The question now becomes, how can we determine the molecular function that these proteins play or the biological processes that they may be a part of? Experimentally, efforts have been made to infer a relationship between *PUFs* and proteins of known function, otherwise known as *PKFs*. This has been done through intentionally expressing a specific gene with byproducts of *PKFs* while measuring which *PUFs* are co-expressed. Although this does not conclusively give us information about *PUFs*, it allows for the association of these PUF genes with biological processes of known function. The goal is to computationally arrive at an association between genes of unknown function and genes of known function. Predicting links between entities in the Gene Ontology could potentially allow us to infer the function that specific proteins play in the body that are not fully understood. We tackle this problem by exploring existing algorithms that deal with unsound queries. We use Neural Theorem Provers (NTP)[9], a path link prediction method that operates on subsymbolic representations for constructing neural networks that can be used to prove queries to a KB and MINERVA[2], and approach that utilizes reinforcement learning for automated reasoning on large knowledge bases, and compare the trade-offs of both on the task at hand.

II. Dataset

We run the NTP and MINERVA algorithms on the Gene Ontology knowledge base. This ontology contains biological

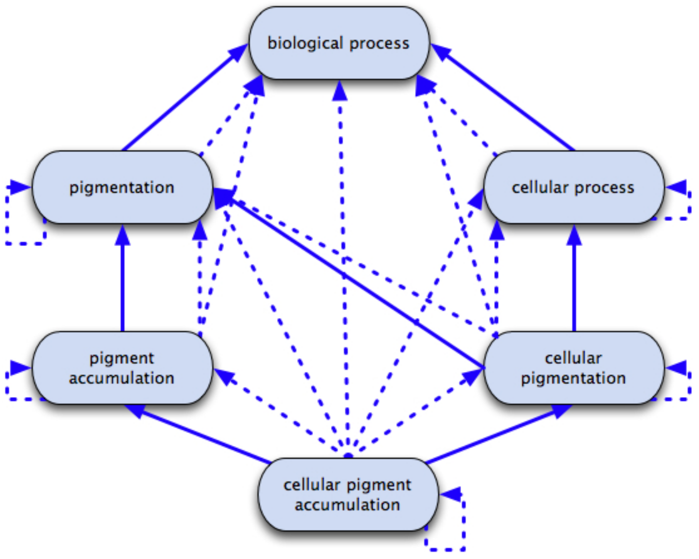
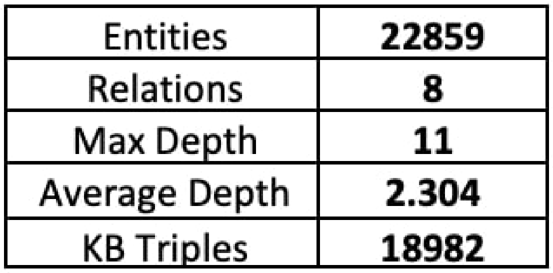


Fig. 1. This what a section extracted from the Gene Ontology may look like. Entities in the GO ontology are connected through relations such as *part-of*, *has-part*, *positively regulates*, *negatively regulates*, and more.

terms such as proteins, genes, and biological functions, and their relationships as explained above.

This table summarizes basic statistics regarding the Gene Ontology Knowledge Graph:



Entities	**22859**
Relations	**8**
Max Depth	**11**
Average Depth	**2.304**
KB Triples	**18982**

Fig. 2. The max depth refers to the longest path from a start entity to an end entity traversing the relations in the graph.

All possible relations that exist in the knowledge graph include *part-of*, *has-part*, *positively-regulates*, *negatively-regulates*, *regulates*, *occurs-in*, *happens-during*, *ends-during*. We utilize the *part-of* to introduce a new relation not originally in the knowledge graph. By introducing this new relation, we hope to derive an inference and predict a path link conditioned We do so through considering the set of all entities e_2 such e_1 relates to e_2 by relation $has-part$, for some entity e_1. We can symbolize this set as: $E_1 = \{e_2|(e_1, r, e_2)\}$. For all entities in our knowledge graph, we obtain the set E_i and determine which entities j satisfy $E_i \cap E_j \neq \emptyset$. For any entities e_1 and e_2 that satisfy this, we introduce the entry in the knowledge graph of $(e_1, similar-parts, e_2)$ such that entity e_1 relates to entity e_2 by relation $similar-parts$. Given the simple

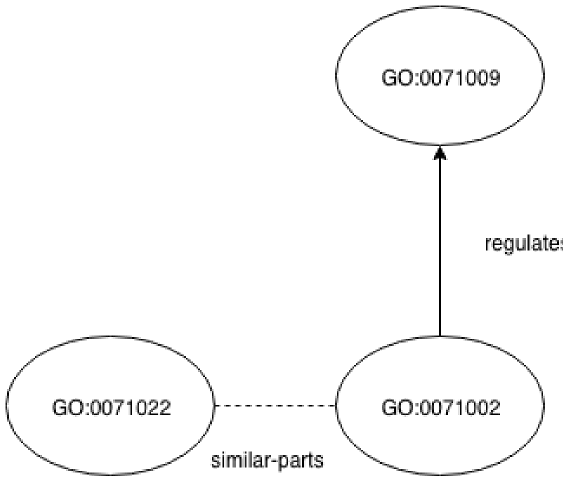


Fig. 3. A simple section from the Gene Ontology. The entities are depicted by their GO IDs. The bold line refers to a relation that exists in the knowledge graph. The dotted line refers to the relation we introduced.

knowledge graph in figure 3, a sample query would be "Does GO:0071022 regulate GO:0071009?". Note that there is no direct link from *GO:0071022* to *GO:0071009* and that we introduced the relation that *GO:0071022* has similar parts to *GO:0071002*. In plain English the query would be: "Does the U12-type post-spliceosomal complex component regulate the U4atac/U6atac x U5 tri-snRNP complex component?"

A. Data Processing

The Gene Ontology does not exist in a knowledge graph format. To properly format it for our means, we utilize the available *.obo* and *.owl* files for the ontology, parsed and produced all triples that comprise the knowledge graph given the relations for each entity. Afterwards we introduced the *similar-parts* relation through the process outlined above and included these in the knowledge graph. Finally, we developed sensible queries through first constructing all possible paths from each entity through to the root entity (since the Gene Ontology is by nature a directed acyclic graph). We thereby constructed sound queries from these paths. Take for example this region from the constructed knowledge graph in figure 4. A sound query we would construct from entries in the knowledge base would be "Does GO:0071035 positively regulate GO:0070924?". However, we concern ourselves with only unsound queries. To construct one such query, observe entities that are related to GO:0071035 by the similar parts relation, and reformulate the query as "Does GO:0071022 positively regulate GO:0070924?"

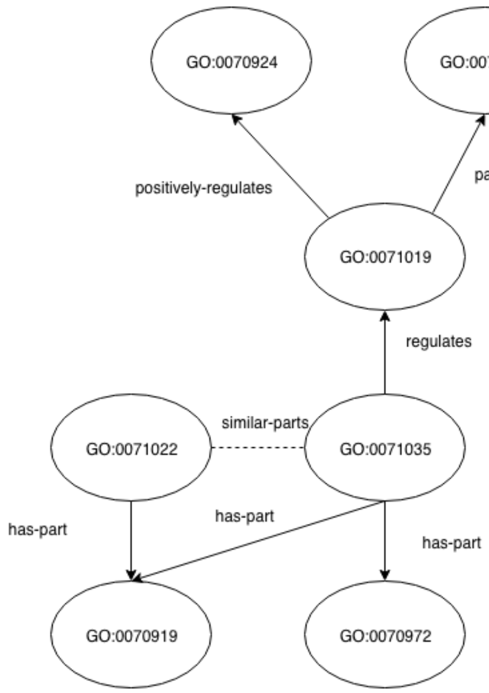


Fig. 4. A more involved section from the Gene Ontology. The entities are depicted by their GO IDs. We construct a sound query from the bold paths and an unsound query through replacing the start entity of the initial query with the related entity.

III. System design

Below we will outline the existing algorithms we aim to reproduce. Both are link prediction over knowledge base algorithms. In addition to an explanation of both approaches, we will emphasize the advantages and disadvantages of each, with the intent to remedy the disadvantages. Prior to an explanation of the algorithms, we must first define what a knowledge graph is. Formally, the knowledge graph $\mathcal{G}$ is a set of triples where each triple is is composed of two entities and a relation and can be depicted as $(e_1, r_2, e_2) \in \mathcal{G}$ [4].

A. Neural Theorem Provers (NTP) Algorithm

1) Overview: Neural Theorem Provers (NTPs)[9] are End-to-end differentiable provers for basic theorems formulated as queries to a KB that use Prologs backward chaining algorithm as a recipe for recursively constructing neural networks that are capable of proving queries to a KB using subsymbolic representations. The success score of such proofs is differentiable with respect to vector representations of symbols. NTPs learn to place representations of similar symbols in close proximity in a vector space and to **induce rules** given prior assumptions about the structure of logical relationships in a KB such as transitivity. Furthermore, NTPs can seamlessly reason with provided domain-specific rules. NTPs demonstrate a high degree of interpretability as they induce latent rules that can be decoded to human-readable symbolic rules.

2) Methodology: NTP uses prolog's backward chaining to translate a query into subqueries via rules. It attempts this for all rules in the knowledge base and thus specifies a depth-first search. On a high level, backward chaining is based on two

functions called *OR* and *AND*. *OR* iterates through all rules (including rules with an empty body, i.e., facts) in a KB and unifies the goal with the respective rule head, thereby updating a substitution set. It is called *OR* since any successful proof suffices (disjunction). If unification succeeds, *OR* calls *AND* to prove all atoms (subgoals) in the body of the rule. To prove subgoals of a rule body, *AND* first applies substitutions to the first atom that is then proven by again calling *OR*, before proving the remaining subgoals by recursively calling *AND*. This function is called *AND* as all atoms in the body need to be proven together (conjunction). As an example, a rule such as *[grandfatherOf, X, Y] : [[fatherOf, X, Z], [parentOf, Z, Y]]* is used in *OR* for translating a goal like *[grandfatherOf, Q, BART]* into subgoals *[fatherOf, Q, Z]* and *[parentOf, Z, BART]* that are subsequently proven by *AND*.
Unification of two atoms, e.g., a goal that we want to prove and a rule head, is a central operation in backward chaining. Two non-variable symbols (predicates or constants) are checked for equality and the proof can be aborted if this check fails. However, we want to be able to apply rules even if symbols in the goal and head are not equal but similar in meaning (e.g. grandfatherOf and grandpaOf) and thus replace symbolic comparison with a computation that measures the similarity of both symbols in a vector space.[9].
So once the query has several proof paths, the network uses a max pooling operation to select the most convincing proof path. Loss is the negative log likelyhood w.r.t target proof success Network is trained end-end using SGD. Vectors are learn such that proof success is high for known facts and low for sampled negative facts.
NTPs are also used for **Inductive Logic Programming** by gradient descent instead of a combinatorial search over the space of rules as, for example, done by the First Order Inductive Learner (FOIL) [8]. Specifically, we are using the concept of learning from entailment [7] to induce rules that let us prove known ground atoms, but that do not give high proof success scores to sampled unknown ground atoms. These are called parameterized rules as the corresponding predicates are unknown and their representations are learned from data. Such a rule can be used for proofs at training and test time in the same way as any other given rule. During training, the predicate representations of parameterized rules are optimized jointly with all other subsymbolic representations. Thus, the model can adapt parameterized rules such that proofs for known facts succeed while proofs for sampled unknown ground atoms fail, thereby inducing rules of predefined structures like the one in figure 4.

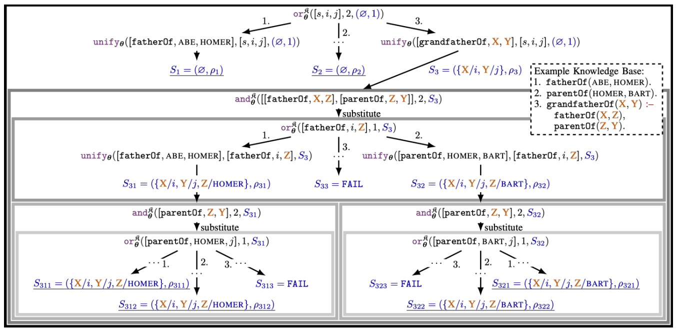

Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :– fatherOf(X, Z), parentOf(Z, Y).
substitute
FAIL

Fig. 5. Exemplary construction of an NTP computation graph for a toy knowledge base. Indices on arrows correspond to application of the respective KB rule. Proof states (blue) are subscripted with the sequence of indices of the rules that were applied. Underlined proof states are aggregated to obtain the final proof success. Boxes visualize instantiations of modules (omitted for unify). The proofs S33, S313 and S323 fail due to cycle-detection (the same rule cannot be applied twice).

B. MINERVA Algorithm

1) Overview: MINERVA is a Reinforcement Learning agent which answers queries in a knowledge graph of entities and relations. Starting from an entity node, MINERVA learns to navigate the graph conditioned on the input query till it reaches the answer entity using a neural reinforcement learning approach. Minerva algorithm begins by representing the environment as a deterministic Markov decision process on a knowledge graph G derived from the KB (KB is a collection of facts stored as triplets $(e1, r, e2)$ where $e1, e2 \in E$ and $r \in R$. Minervas RL agent is given an input query of the form $(e1_q, r_q, ?)$.

Starting from vertex corresponding to $e1_q$ in the knowledge graph G, the agent learns to traverse the environment/graph to mine the answer and stop when it determines the answer. The agent is trained using policy gradient more specifically by REINFORCE [10].

2) Methodology: The environment is a finite horizon, deterministic and partially observed Markov decision process that lies on a knowledge graph derived from the KB. Formally a knowledge graph is a directed labeled multigraph *G = (V,E,R)* , where *V* and *E* denote the vertices and edges of the graph respectively. On this graph the deterministic partially observable Markov decision process, which is a 5-tuple (S, O, A, δ, R), is described as below:

- *States*: The state space S consists of all possible query-answers cartesian product with the set of entities. Intuitively, we want a state to encode the query $(e1_q, r_q)$, the answer $(e2_q)$, and a location of exploration e_t (current node of the entity). Thus overall a state S S is represented by $S = (e_t, e1_q, r_q, e2_q)$ and the state space consists of all valid combination
- *Observations*. The complete state of the environment is not observable, but only its current location of exploration and query can be observed but not the answer.
- *Actions*. The set of possible actions A_S from a state $S = (e_t, e1_q, r_q, e2_q)$ consists of all outgoing edges of the vertex e_t in *G*.
- *Transition*. The environment evolves deterministically by just updating the state to the new vertex pointed by the edge selected by the agent through its action. The query and answer remains the same.
- *Rewards*. There is only a terminal reward of +1 if the current location is the correct answer at the end and 0 otherwise

Policy Network is randomized history-dependent policy which is just the sequence of observations and actions taken. An agent based on LSTM encodes the history H_t as a continuous vector. Based on the history embedding h_t , the policy network makes the decision to choose an action from all available actions (A_{St}) conditioned on the query relation. The network taking these as inputs is parameterized as a two-layer feedforward network with ReLU nonlinearity which takes in the current history representation h_t and the embedding for the query relation r_q and outputs a probability distribution over the possible actions from which a discrete action is sampled.

Training : For the policy network described above, we want to find parameters θ that maximizes the expected reward. To solve this optimization problem REINFORCE [10] is applied as below:

- The first expectation is replaced with empirical average over the training dataset.
- For the second expectation, is approximated by running multiple rollouts for each training
- For variance reduction, a common strategy is to use an additive control variate baseline [3]. A moving average of the cumulative discounted reward as the baseline is used. The weight of this moving average as a hyperparameter is tuned.
- To encourage the policy to sample more diverse paths rather than sticking with a few, an entropy regularization term is added to the cost function after multiplying it by a constant (β). β isb treated as a hyperparameter to control the exploration exploitation trade-off.

IV. Experiments

Consistent with previous work in the domain, we carry out experiments on our Gene Ontology knowledge base and compare the results between the baseline implementation with Neural Theorem Provers (NTP)[9] and MINERVA[2] and compare HITS@m [1]. As previously mentioned the GOKB is a large scale knowledge base with 22859 entities and 8 relations. The max depth of to reach a answer node can be upto 11 whereas are the average depth is around 2. Which is one of the reaons the hyperparameters selected are taking into consideration these parameters of the knowledge graph. We have conciously decided not to select an emsemble approach as we wanted to see the results of the performance of a single model and comapare it with its equivalent baseline. The experiments were run on a TITANX GPU using Tensorflow.

Finally, to note one point of consideration that MINERVA utilized in predicting path links. The path that was chosen was heavily impacted by the input query. With regards to the Gene Ontology knowledge graph, this weight assigned to particular paths allowed for more sensible paths. To illustrate this point, consider the subsection of the GO knowledge graph in figure 5. Consider the test query: ”Does GO:0071011 regulate GO:0070918?” As the algorithm traverses a path and reaches the entity GO:0071116, it can choose between continuing along the ”part-of” relation or the ”happens-during” relation. MINERVA assigns a much higher probability of taking the edge ”part-of” than ”happens-during” since it less likely that the entity correlating to when GO:0071116 happens during will in turn regulate any other entity.

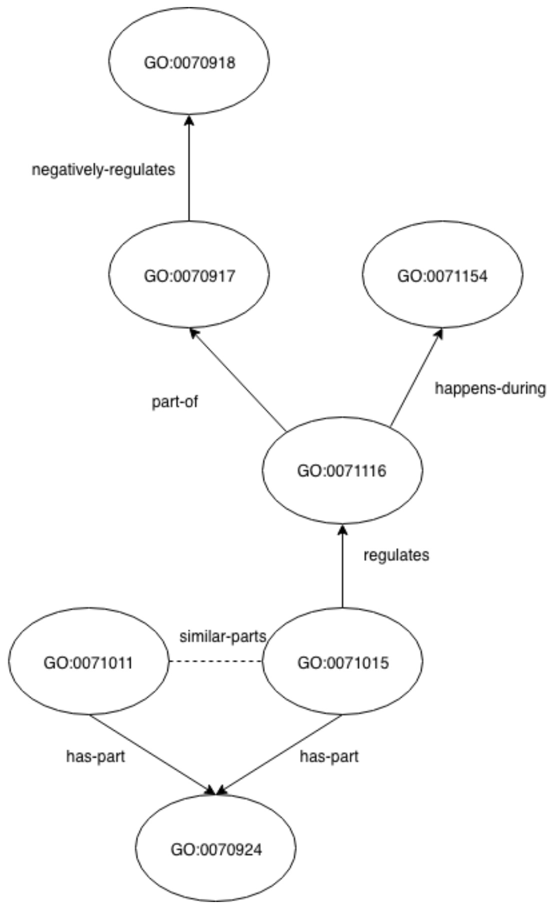


Fig. 6. A more involved section from the Gene Ontology. The entities are depicted by their GO IDs. In this case, one path may be preferred in the MINERVA algorithm (outlined below) over another.

V. Results and Discussion

Results for the different model variants on the benchmark GOKB KBs are shown in Table II. The run times of NTP compared to MINERVA were drastically different. We could only run 1000 iterations of NTP in about 6 hours, whereas the MINERVA model completed 2000 iterations in an hour. NTPs suffers from severe computational limitations since the neural network is representing all possible proofs up to some predefined depth, since a score can be computed between any two vectors, the computation graph becomes quite large because of such soft-matching during substitution step of backward chaining. . MINERVA on the other hand is efficient at inference time since it has to essentially search for answer entities in its local neighborhood, whereas previous methods rank all the entities in the dataset. As mentioned previously on the our dataset of GOKB, the wall clock running time of MINERVA is **33 iterations/minute** whereas that of a GPU implementation of NTP is a mere 2.7 iterations a minute.

Model	Hits@3	Hits@5
NTP	0.38	0.41
MINERVA	**0.56**	**0.64**

TABLE I
BENCHMARK RESULTS FOR GO KB ON NTP AND MINERVA

Hyper Parameter	**Value**	**Hyper Parameter**	**Value**
total_iterations	2000	total_iterations	1000
path_length	3	input_size	100
hidden_size	100	l2	0.001
embedding_size	100	max_depth	3
beta	0.05	epsilon	1e-10
lambda	0.05	lr	0.001
lstm_layers	3	optimizer	Adam
batch_size	256	unification	cos
pool	max	ensemble	false

TABLE II
HYPER PARAMETERS FOR MINERVA (LEFT) AND NTP (RIGHT)

Also as observed MINERVA outperforms NTP on the GOKB on both hits@3 and hits@5 metrics as shown in Table II for unsound queries. This is mostly attributed to the below:

- **Effectiveness of Remembering Path History**. MINERVA encodes the history of decisions it has taken in the past using LSTMs.
- **NO-OP and Inverse Relations**. At each step, MINERVA can choose to take a NO-OP edge and remain at the same node. This gives the agent the flexibility of taking paths of variable lengths. Some questions are easier to answer than others and require lesser steps of reasoning and if the agent reaches the answer early, it can choose to remain there.
- **Query based Decision Making**. At each step before making a decision, our agent conditions on the query relation.

This can also be seen from the figure 6 below where we see rules and paths MINERVA can learn. This example is on a standard dataset.

(i) **Can learn general rules:**

(S1) LocatedIn(X, Y) $\leftarrow$ LocatedIn(X, Z) & LocatedIn(Z, Y)
(S2) LocatedIn(X, Y) $\leftarrow$ NeighborOf(X, Z) & LocatedIn(Z, Y)
(S3) LocatedIn(X, Y) $\leftarrow$ NeighborOf(X, Z) & NeighborOf(Z, W) & LocatedIn(W, Y)

(ii) **Can learn shorter path:** Richard F. Velky $\xrightarrow{\text{WorksFor}}$?

Richard F. Velky $\xrightarrow{\text{PersonLeadsOrg}}$ Schaghticokes $\xrightarrow{\text{NO-OP}}$ Schaghticokes $\xrightarrow{\text{NO-OP}}$ Schaghticokes

(iii) **Can recover from mistakes:** Donald Graham $\xrightarrow{\text{WorksFor}}$?

Donald Graham $\xrightarrow{\text{OrgTerminatedPerson}}$ TNT Post $\xrightarrow{\text{OrgTerminatedPerson}^{-1}}$ Donald Graham $\xrightarrow{\text{OrgHiredPerson}}$ Wash Post

Fig. 7. A few example of paths found by MINERVA on the COUNTRIES and NELL. MINERVA can learn general rules as required by the COUNTRIES dataset (example (i)). It can learn shorter paths if necessary (example (ii)) and has the ability to correct a previously taken decision (example (iii)).

VI. CONCLUSION AND FUTURE WORK

We explored automated reasoning for unsound queries on a new dataset *GOKB* and compared its inductive reasoning performance on both Neural Theorem Provers (NTP) and MINERVA. We observed significantly better performance on both *hits@3* and *hits@5* benchmarks on the MINERVA dataset. The key for MINERVA to outperform prior efforts like NTP is that it uses reinforcement learning approach which learns how to navigate the graph conditioned on the input query to find predictive paths where the goal is to take an optimal sequence of decisions (choices of relation edges) to maximize the expected reward (reaching the correct answer node). A drawback of MINERVA could be that it only has a 0 and 1 reward, i.e. the agent gets rewarded only when the correct target entity is reached, which overlooks partial path correctness. In the future we want to look into reinforcement learning approaches where the agent can be rewarded for partial path correctness using some form of reward shaping techniques.[6]

REFERENCES

[1] Alberto Garca-Durn Jason Weston Antoine Bordes, Nicolas Usunier and Oksana Yakhnenko. In advances in neural information processing systems. 2013.

[2] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alexander J. Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning, 2017.

[3] John Hammersley et. al. Monte carlo methods. *Springer Science Business Media*, 2013.

[4] Michael Frber, Basil Ell, Carsten Menne, Achim Rettinger, and Frederic Bartscherer. Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago. *Semantic Web 1*, 2016.

[5] K. Horan, C. Jang, J. Bailey-Serres, R. Mittler, C. Shelton, J. F. Harper, J.-K. Zhu, J. C. Cushman, M. Gollery, T. Girke, and et al. Annotating genes of known and unknown function by large-scale coexpression analysis. *Plant Physiology*, 147(1):4157, 2008.

[6] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. *CoRR*, abs/1808.10568, 2018.

[7] Stephen Muggleton. Inductive logic programming. 1991.

[8] J. Ross Quinlan. Learning logical definitions from relations. 1990.

[9] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3791–3803, 2017.

[10] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement Learning*, page 532, 1992.