Logic Programming for Knowledge Graph Completion

Damiano Azzolini, Matteo Bonato, Elisabetta Gentili, and Fabrizio Riguzzi
University of Ferrara June 27, 2024

CILC 2024

Contribution

Question: can parameter learning for Probabilistic Logic Programming be a competitive algorithm to solve the knowledge graph completion (KGC) task?

Answer: (probably) no.

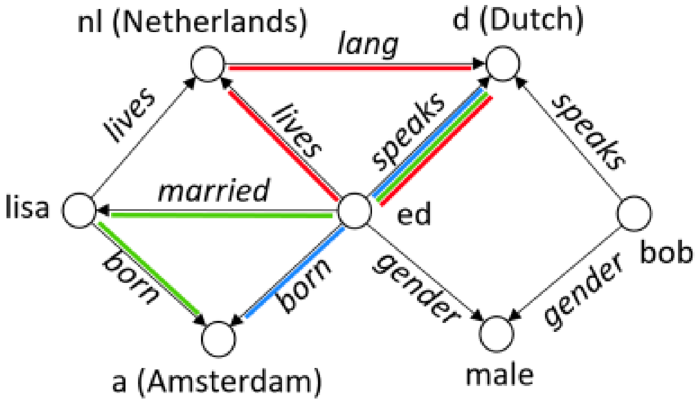
Knowledge Graphs I

Knowledge Graphs (KGs) are graph-based representations of knowledge in terms of relationships between entities.

A KG can be represented as a set of triples (h, r, t) where:

- r is the relation
- h and t are the head (start) and tail (end) entities of the relation

Knowledge Graphs II



From AnyBURL

Knowledge Graphs III

Real-world KGs are usually incomplete and sparse, thus inference of missing information (entities or relationships) is often required.

This task is referred to as knowledge graph completion (KGC).

Knowledge Graphs IV

We focus on the link prediction task: predicting either the tail t of a triple $(h, r, ?)$, or the head h of a triple $(?, r, t)$.

In other words, given a completion task $(h, r, ?)$ on a graph $\mathbb{G}$, the goal is to find an entity t such that $(h, r, t) \notin \mathbb{G}$ is true.

Knowledge Graphs V

Candidates triple are associated with a score, sorted in descending order, and ranked.

Different approaches to break ties: minimum, maximum, average (i.e., it assigns the average rank of the group to each candidate – we adopted this), etc, ranking.

Example: t_0 score 10, t_1 score 2, and t_2 score 2. If we apply average rank we have:

- t_0 rank 1
- t_1 rank $(2+3)/2 = 2.5$
- t_2 rank $(2+3)/2 = 2.5$

Knowledge Graphs VI

The graph $\mathbb{G}$ is often split into three datasets, training set, test set, and validation set, used to train the model and evaluate its performance.

Given a set of test triples, KGC algorithms are usually evaluated on link prediction tasks in terms of the following metrics:

- Mean Rank (MR): average rank of the test triples
- Mean Reciprocal Rank (MRR): average reciprocal individual rank of the test triples
- Hits@K: represents the proportion of the test triples ranked in the top K positions

Knowledge Graphs VII

Candidates for a triple may not be present in the test set but they may appear in the training or validation set.

To not penalize the ranking of the correct candidate, predictions that appear in triples of the training or validation set can be filtered out: filtered metrics.

KGC Tools - AnyBURL I

Iteratively samples from a KG random paths of length n, where n is the number of edges and starts from $n = 2$.

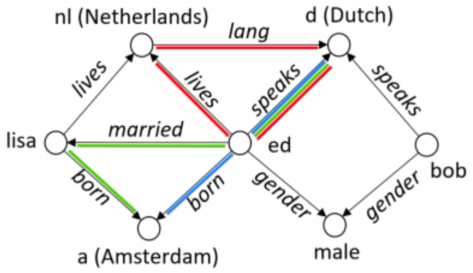
This process continues until a certain saturation parameter (which depends on the contribution to the overall performance of the considered rules) is reached.

KGC Tools - AnyBURL II

These paths are then assembled into ground logical rules where the first edge is considered as the head and the remaining as the body.

Given a set of triples $\{(e_0, h_0, e_1), (e_1, b_1, e_2), \ldots, (e_n, b_n, e_{n+1})\}$, we can construct a ground path rule R of the form
$h(e_0, e_1) \leftarrow b_1(e_1, e_2), \ldots, b_n(e_n, e_{n+1})$.

KGC Tools - AnyBURL III



Green: $married(el, lisa) \leftarrow born(lisa, amsterdam)$

KGC Tools - AnyBURL IV

AnyBURL extracts three types of rules.

Name	Structure
C	$h(Y, X) \leftarrow b_1(X, A_2), \ldots, b_n(A_n, Y)$
AC1	$h(e_0, X) \leftarrow b_1(X, A_2), \ldots, b_n(A_n, e_{n+1})$
AC2	$h(e_0, X) \leftarrow b_1(X, A_2), \ldots, b_n(A_n, A_{n+1})$

X and Y are variables that appear in the head, while A_i can appear only in the body. Lowercase letters indicate constants.

University of Ferrara

KGC Tools - AnyBURL V

The score of a rule R is its confidence $conf(R)$, calculated as the fraction of body groundings that result in a correct head grounding based on the training data.

KGC Tools - AnyBURL VI

When the number of rules is high, computing KGC metrics is very expensive.

AnyBURL limits the search for candidates if during the grounding of a rule a branch with more than 10^4 children is found.

KGC Tools - AnyBURL VII

For example, consider the rule

$$hasGender(X, Y) \leftarrow bornIn(X, A), bornIn(B, A), hasGender(B, Y)$$

and the query $hasGender(susi, T)$ where *susi* is born in the USA. The second body atom unifies B with each person known to be born in the USA, so there can be more than 10^4 instantiations. In this case, AbyBURL only retrieves the first 10^4.

This only provides an approximation of the metrics but it is a good trade-off between speed and precision.

KGC Tools - LIFTCOVER+ I

Liftable PLP programs contain clauses with a single annotated atom h_i in the head annotated with a probability Π_i and the predicate of this atom is the same for all clauses:

$$h_i : \Pi_i :- b_{i1}, \ldots, b_{iu_i}$$

KGC Tools - LIFTCOVER+ II

$$married(A, B) : 0.9 :- born(A, C), lives(B, C).$$

$$married(A, B) : 0.7 :- speaks(A, C), studied(B, C).$$

KGC Tools - LIFTCOVER+ III

To compute the probability of a query q, it is necessary to find only the number of ground instantiations m_i of each of the n clauses so that the body is true and the head is equal to q.

$$P(q) = 1 - \prod_{i=1}^{n} (1 - \Pi_i)^{m_i}$$

KGC Tools - LIFTCOVER+ IV

The parameter learning task consists in learning the probabilities of the rules such that the likelihood of a set of examples is maximized.

- LIFTCOVER adopts Expectation Maximization or Limited-memory BFGS (LBFGS)
- LIFTCOVER+ adopts Expectation Maximization and regularization to prevent overfitting or gradient descent

For both, clauses with a low probability value (less than a user defined parameter ϵ) are discarded.

KGC Tools - LIFTCOVER+ V

Positive and negative examples should be provided.

We can solve the KGC task with LIFTCOVER+ by generating rules and then appying its parameter learning algorithm.

Rules Generation Methods I

Three generation methods

- generation of cyclic rules
- generation of rules as in AnyBURL
- weighted sampling

Rules Generation Methods II

Generation of cyclic rules (implemented in SWI-Prolog): generate only cyclic rules by generating tuples of relations $r_0, r_1, \ldots,$ and then converting them to rules.

Each triple (h, rel, t) in the dataset is translated into an atom $t(h, rel, t)$. For example, the tuple `(r0,r1,r2,r3)` is translated into the rule

```
r(A,r0,B):- r(A,r1,C),r(C,r2,D),r(D,r3,B).
```

where $r/3$ is defined as

```
r(S,R,T):- t(S,R,T).
r(S,i(R),T):- t(T,R,S).
```

Rules Generation Methods III

The tuples of relations are obtained by starting from triples in the training set. For example, if we want to extract 4 relations we look for values of `R1,R2,R3,R4` such that the query

```
r(h,R1,C),r(C,R2,D),r(D,R3,E),r(E,R4,t).
```

succeeds at least once. Then duplicate tuples are removed.

Removing duplicated tuples is very expensive.

Rules Generation Methods IV

Generating Rules by Weighted Sampling (implemented in SWI-Prolog)

- we associate each relation with a probability proportional to its occurrences
- we first sample a relation from L_r to consider as head relation
- we select a random number between two and four and sample again the same number of relations from L_r, that will be considered for the body

Both sampling phases take into account the probability associated with each relation.

Rules Generation Methods V

We generate rules by creating an atom with two variables for each relation and then connect the atoms to create a chain rule.

Example: suppose we have *r*0 for the head and *r*1 and *r*2 for the body. We obtain the rule

```
r(A,r0,B) :- r(B,r1,C), r(C,r2,D)
```

Once we have all the rules, we compute the cumulative number of instantiations for all the bodies, call it n_i, and associate each rule with a score given by the ratio between its number of body instantiations and n_i.

Experimental Evaluation I

Selected datasets

Dataset	Train	Valid	Test	Total	Entities	Relations
Nations	1,592	199	201	1'992	28	55
WN18RR	86,835	3,034	3,134	93,003	71,453	11
FB15k-237	272,115	17,535	20,466	310,116	14,505	237
Nell	228,426	129,810	144,307	502,543	63,361	400

Experimental Evaluation II

Maximum length of the body of rules and number of rules for each dataset and each rule generation method.

Dataset	Rule Generation Method	Max Length	#Rules
Nations	Paths	3	305,365
	AC1 + AC2 + C	3	2,243
	Weighted Sampling	4	10^5
WN18RR	Paths	3	3,076
	Weighted Sampling	4	10^5
FB15K-237	Paths	3	33,696
	Weighted Sampling	4	10^5
Nell	Paths	3	101,242
	Weighted Sampling	4	10^5

Experimental Evaluation III

We adopted the EM algorithm for parameter learning. It was performed on the Leonardo HPC system of Cineca, using machines with

- Intel Xeon Platinum 8358
- 2.60GHz
- 32 cores
- 481GB of RAM
- Nvidia A100 GPUs with 64GB of memory

The Weighted Sampling algorithm was run on a machine running at 2.40 GHz with 16 GB of RAM with a time limit of 8 hours (this approach does not have a learning phase).

Experimental Evaluation IV

We implemented the metrics computation in SWI-Prolog.

Tested the following configurations:

- Sampling path + EM
- Sampling path + confidence as score for rules
- AC1 + AC2 + C + EM
- Weighted Sampling

Experimental Evaluation V

Dataset	Approach	H@1	H@3	H@5	H@10	MRR
Nations	Paths+EM	0.457	0.786	0.860	**0.995**	0.638
	Paths+Conf	0.492	**0.791**	**0.855**	0.970	**0.658**
	AC1+AC2+C+EM	0.218	0.572	0.761	0.910	0.440
	W. S.	**0.507**	0.781	0.845	0.965	0.647
WN18RR	Paths+EM	0.340	0.453	0.493	0.537	0.419
	Paths+Conf	**0.437**	**0.505**	**0.537**	**0.573**	**0.489**
	W. S.	0.030	0.082	0.110	0.151	0.074
FB15K-237	Paths+EM	0.214	0.309	0.353	0.426	0.284
	Paths+Conf	**0.244**	**0.352**	**0.403**	**0.480**	**0.323**
Nell	Paths+EM	0.001	0.001	0.002	0.005	0.001
	Paths+Conf	**0.001**	**0.002**	**0.003**	**0.006**	**0.003**

University of Ferrara

Conclusions

- Sampling paths and applying EM is not beneficial maybe due to the syntethic generation of negative examples
- Weighted sampling cannot handle larger datasets due to the high number of instantiations for rules
- Generating the three types of rules as in AnyBULR and then applying EM is not feasible since too much memory is required in the learning phase
- Metrics computation in SWI-Prolog is very slow (maybe unification has too much overhead in answering such simple queries)