MLN4KB: an efficient Markov logic network engine for large-scale knowledge bases and structured logic rules

Huang Fang
Cognitive Computing Lab, Baidu Research
No.10 Xibeiwang East Road, Beijing 100193, China
`fanghuang@baidu.com`

Yang Liu
Cognitive Computing Lab, Baidu Research
No.10 Xibeiwang East Road, Beijing 100193, China
`liuyang173@baidu.com`

Yunfeng Cai
Cognitive Computing Lab, Baidu Research
No.10 Xibeiwang East Road, Beijing 100193, China
`caiyunfeng@baidu.com`

Mingming Sun
Cognitive Computing Lab, Baidu Research
No.10 Xibeiwang East Road, Beijing 100193, China
`sunmingming01@baidu.com`

ABSTRACT

Markov logic network (MLN) is a powerful statistical modeling framework for probabilistic logic reasoning. Despite the elegancy and effectiveness of MLN, the inference of MLN is known to suffer from an efficiency issue. Even the state-of-the-art MLN engines can not scale to medium-size real-world knowledge bases in the open-world setting, i.e., all unobserved facts in the knowledge base need predictions. In this work, by focusing on a certain class of first-order logic rules that are sufficiently expressive, we develop a highly efficient MLN inference engine called MLN4KB that can leverage the sparsity of knowledge bases. MLN4KB enjoys quite strong theoretical properties; its space and time complexities can be exponentially smaller than existing MLN engines. Experiments on both synthetic and real-world knowledge bases demonstrate the effectiveness of the proposed method. MLN4KB is orders of magnitudes faster (more than 10^3 times faster on some datasets) than existing MLN engines in the open-world setting. Without any approximation tricks, MLN4KB can scale to real-world knowledge bases including WN-18 and YAGO3-10 and achieve decent prediction accuracy without bells and whistles.

We implement MLN4KB as a Julia package called MLN4KB.jl. The package supports both maximum a posteriori (MAP) inference and learning the weights of rules. MLN4KB.jl is public available at `https://github.com/baidu-research/MLN4KB`.

KEYWORDS

Markov logic network, knowledge graph completion.

ACM Reference Format:
Huang Fang, Yang Liu, Yunfeng Cai, and Mingming Sun. 2023. MLN4KB: an efficient Markov logic network engine for large-scale knowledge bases and structured logic rules. In *Proceedings of the ACM Web Conference 2023 (WWW '23), May 1–5, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3543507.3583248

1 INTRODUCTION

Relational data and knowledge bases link real-world entities such as people, events, and words with diverse relations. Nowadays, almost all institutes, companies, websites, and mobile apps generate and manipulate relational data everyday. Discovering values from massive relational data stays at the core of data science and has been an active research topic in both industry and academia.

Due to the prevalence of relational data, logic and probabilistic reasoning over relational data and knowledge bases has been an important subfield of artificial intelligence (AI). Many modern AI applications such as question answering, information extraction, semantic parsing, and social network analysis all rely on reasoning over relational data in large knowledge bases. Some communities even believe that building a proper relation (concepts) framework and logical reasoning system is the key to imitating human recognition. Among many logic reasoning approaches, the Markov logic network (MLN) is a simple and flexible reasoning framework that unifies the classic first-order logic and probabilistic graphical model; see Figure 1 for an illustration of an MLN. In short, MLN defines a probability distribution over a set of relational data via some first-order logic rules; MLN assigns higher probability to worlds that satisfy more grounded rules, and vice versa (c.f. Section 3). MLN has received substantial interest over the last two decades since its first appearance. On the theory side, MLN enjoys simple formulation (eq. (3)) and has the ability to express all probability distribution over discrete or finite-precision random variables [8, Theorem 2.5]. On the empirical side, MLN has reported state-of-the-art performance for a wide range of applications such as collective classification [34], link prediction [31], semantic parsing [35], etc.

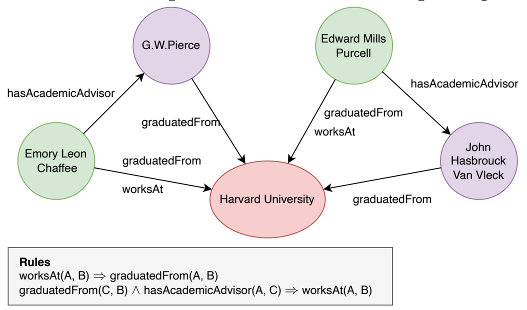

Figure 1: An illustration of MLN, the knowledge graph is a subset extracted from the YAGO3-10 dataset

Despite the elegancy and empirical success of MLN, MLN has an efficiency issue; both the inference and learning of MLN are intractable in general. Huge efforts have been devoted to increasing the scalability of MLN either from the engineering or algorithmic perspectives. For examples, Tuffy [32] and DeepDive [40, 42] leverage techniques from relational database management systems (RDBMS) and develop efficient local searching and sampling methods using a RDBMS; PSL [1] adopts continuous relaxation and borrows techniques from continuous optimization to speedup the inference and learning of MLN. Despite the notable progress made by pioneering researchers, current MLN software still cannot scale to knowledge bases with hundreds of thousands of entities.

The state-of-the-art MLN software almost exclusively divide the inference (or learning) into two phases: the *grounding* and *searching* phases. The grounding phase grounds given first-order logic rules into clauses and forms a large weighted SAT problem; the *searching* phase then solves the constructed SAT problem via some optimization subroutines. For medium and large datasets, the grounding phase is usually more computationally intensive than the searching phase and is the bottleneck of existing MLN software. In this work, we argue that *the flexibility of the current MLN solvers limits their scalability.* By narrowing down the scope of first-order logic rules and considering logic rules with certain structures (c.f. Assumption 3.1), we develop an efficient MLN inference engine termed MLN4KB that can exploit the sparsity of large knowledge bases and mitigate the computational overhead from the grounding phase. We show that MLN4KB is more time and space efficient than the existing MLN inference algorithms both theoretically and empirically. Formally, we summarize our contributions as follows.

- By exploiting the structure of a certain class of logic rules, we develop an efficient MLN inference engine called MLN4KB. MLN4KB is designed for large-scale knowledge bases under the *open-world setting* (Assumption 3.3), it stores only violated clauses in memory and can circumvent the efficiency issue of the grounding phase. The theoretical properties of MLN4KB are given in Section 5.
- To foster future research of MLN, we open-source our implementation of MLN4KB as a Julia package MLN4KB.jl. MLN4KB.jl supports both the inference and learning of MLN.
- Extensive experiments on both synthetic and real-world knowledge bases demonstrate the efficiency of MLN4KB.jl.

2 RELATED WORK

Logic and AI. Whether logic is the fundamental of AI has been a controversial topic from time to time, but most AI researchers would agree that first-order logic and knowledge representation play an important role in at least some central subfields of AI [48]. A huge line of works has been carried on in the second half of the 20th century to study computational logic [14, 23–26, 28]. Various inductive logic programming methods such as Prolog has been developed to perform logic reasoning efficiently [52, 53].

Markov logic network. The success of AI in the past two decades is indispensable to the emergence of statistical learning. Markov logic network [8] is an elegant logic reasoning framework that combines the classic first-order logic and the more advanced probabilistic graphical model. MLN has been extensively studied in the past two decades both theoretically and empirically [3, 8, 9, 27, 55]; we refer interested readers to the review article from Lowd and Domingos [10] for more details. Despite the success of MLN on many real-world applications, both the inference and learning of MLN are NP-hard in general, and therefore MLN cannot scale to large datasets, especially in the open-world setting. Many techniques have been developed in the past two decades to speedup the inference of MLN, including lazy grounding, lifted belief propagation, smart sampling techniques, etc [33, 36, 39, 41, 43, 44, 47]. Based on these techniques, a number of highly-engineered MLN solvers such as Alchemy [22], Tuffy [32], DeepDive [40] and PSL [1] were developed for practitioners to deploy MLN more efficiently. This work also aims at developing a high-performance MLN solver by exploiting the structure of a certain class of first-order logic rules and the sparsity of large knowledge base. At first glance, our method may seems related to the lazy grounding technique [43]. We note that the technique developed in this work is fundamentally different from lazy grounding; see the detailed discussion in Remark 5.1.

Besides MLN, there are other probabilistic logic reasoning frameworks such as the knowledge-based model construction [51], stochastic logic programming [29], probabilistic relational models [12], etc. We focus on MLN in this work and do not include the above reasoning frameworks in the following content.

Embedding-based method. Embedding-based methods have revolutionized certain fields of natural language processing and have achieved state-of-the-art performance for some knowledge graph applications [6, 7, 30, 46, 50]. There are some recent interests in combining knowledge graph embedding and graph neural networks with first-order logic rules to speedup the inference and reduce the storage of MLN [37, 56]. These recent works can be viewed as extensions and approximations of the classic MLN and are tangential to the purpose of this work. Although embedding-based methods such as GNNs have achieved promising performance in many real-world applications. Purely logic-based methods such as MLN still have their unique advantages. For example, logic-based methods can better leverages the knowledge from human experts and are more sample efficient when high-quality rules are provided.

3 PROBLEM SETUP

3.1 Preliminaries

We introduce some basic concepts about knowledge bases and Markov logic networks in this section.

Knowledge base. A knowledge base $\mathcal{K}$ consists of a set of *entities* $\mathcal{E}$ and a set of *relations* (aka. *predicates*) $\mathcal{R}$. Given any pair of entities $(e_1, e_2) \in \mathcal{E} \times \mathcal{E}$, and a relation $r \in \mathcal{R}$, the relation maps the pair of entities to either 1 or 0, i.e., $\mathsf{r} : \mathcal{E} \times \mathcal{E} \rightarrow \{0, 1\}\ \forall \mathsf{r} \in \mathcal{R}$, with the meaning that the head entity e_1 has the relation r with the tail entity e_2 or not. For example, $\mathtt{father}(Bob, Anna) = 1$ indicates that Bob is Anna's father.

In the knowledge base completion problem, people observe a set of triplets (aka. facts) $O = \{(h_i, \mathsf{r}_i, t_i)\}_{i=1}^{n}$ along with their true assignments $\mathbf{O} = \{\mathsf{r}(h, t) \mid (h, \mathsf{r}, t) \in O\}$ (aka. evidences). Denote the unobserved facts as $\mathcal{H} = \mathcal{E} \times \mathcal{R} \times \mathcal{E} \backslash O$. The knowledge graph completion task aims to infer the assignments of all unobserved facts $\mathbf{H} = \{\mathsf{r}(h, t) \mid (h, \mathsf{r}, t) \in \mathcal{H}\}$.

First-order logic rule. A first-order logic rule F associated with a knowledge base $\mathcal{K}$ is a logic expression based on the relations in $\mathcal{K}$. Such a logic expression is a disjunction of literals and each literal can be negated. And in this paper, we only consider the literal of the form $\mathtt{r}(h, t)$. Specifically, the logic rules considered in this paper are all of the form

$$\Big(\vee_{i \in \mathcal{I}_F^-} \; !\mathtt{r}_i(A_i, B_i) \Big) \vee \Big(\vee_{i \in \mathcal{I}_F^+} \mathtt{r}_i(A_i, B_i) \Big), \tag{1}$$

where $\mathcal{I}_F^-$ and $\mathcal{I}_F^+$ are two index sets containing the indices of literals that are negated or not, respectively.

Logic rules in the form (1) are quite expressive. They can be equivalently taken as implications from conditions to consequences:

$$\wedge_{i \in \mathcal{I}_F^-} \mathtt{r}_i(A_i, B_i) \implies \vee_{i \in \mathcal{I}_F^+} \mathtt{r}_i(A_i, B_i). \tag{2}$$

For example, the first order logic rule $!\mathtt{husband}(X, Y) \vee \mathtt{wife}(Y, X)$ is equivalent to $\mathtt{husband}(X, Y) \implies \mathtt{wife}(Y, X)$, meaning that if X is Y's husband, then Y is X's wife; $!\mathtt{father}(X, Y) \vee !\mathtt{brother}(Y, Z) \vee \mathtt{father}(X, Z)$ is equivalent to $\mathtt{father}(X, Y) \wedge \mathtt{brother}(Y, Z) \implies \mathtt{father}(X, Z)$, meaning that if X is Y's father and Y is Z's is brother, then X is Z's father. In addition, using the equivalence between the logic expression (1) and the implication (2), it is easy to see that some commonly used types of logic rules can be written in the form (1): *Composition rules.* $\mathtt{r}_k$ is the composition of $\mathtt{r}_i$ and $\mathtt{r}_j$ if $\mathtt{r}_i(X, Y) \wedge \mathtt{r}_j(Y, Z) \implies \mathtt{r}_k(X, Z)$ $\forall X, Y, Z \in \mathcal{E}$; *Inverse rules.* $\mathtt{r}^{-1}$ is the inverse of $\mathtt{r}$ if $\mathtt{r}(X, Y) \implies \mathtt{r}^{-1}(Y, X)$ $\forall X, Y \in \mathcal{E}$; *Symmetric rules.* $\mathtt{r}$ is symmetric if $\mathtt{r}(X, Y) \iff \mathtt{r}(Y, X)$ $\forall X, Y \in \mathcal{E}$; *Subrelation rules* $\mathtt{r}'$ is a subrelation of $\mathtt{r}$ if $\mathtt{r}(X, Y) \implies \mathtt{r}'(X, Y)$ $\forall X, Y \in \mathcal{E}$.

Markov logic network. A full assignment (aka. grounding or instantiation) to all facts $\{\mathtt{r}(h, t) \mid \forall \mathtt{r} \in \mathcal{R}, h, t \in \mathcal{E}\}$ is called a *possible world*; the assignment to both the observed facts and the unobserved facts $\mathbf{O} \cup \mathbf{H}$ determines a possible world. Given m first-order logic rules $\mathcal{F} = \{F_i\}_{i=1}^m$, the Markov logic network encodes the possible world $\mathbf{O} \cup \mathbf{H}$ as a random variable and defines its distribution as

$$\Pr[\mathbf{O}, \mathbf{H}] \propto \exp\left(\sum_{i=1}^m w_i l_i(\mathbf{O}, \mathbf{H}) \right) \propto \exp\left(- \sum_{i=1}^m w_i n_i(\mathbf{O}, \mathbf{H}) \right), \tag{3}$$

where $w_i \in \mathbb{R}$ is the weight associated with the i-th logic rule, $l_i(\mathbf{O}, \mathbf{H})$ and $n_i(\mathbf{O}, \mathbf{H})$ are the number of times that the i-th logic rule is satisfied and violated under the world $\mathbf{O} \cup \mathbf{H}$ respectively. We refer interested readers to Domingos and Lowd [8] for more details about the definition of MLNs.

3.2 Assumptions and Notations

We state the key assumptions we made in this work.

Assumption 3.1 (Structured logic rules). *Given a rule $F \in \mathcal{F}$ of the form* (1)*, we assume that* $\cup_{i \in \mathcal{I}_F^-} \{A_i, B_i\} \supseteq \cup_{i \in \mathcal{I}_F^+} \{A_i, B_i\}$.

The above assumption narrows the scope of logic rules considered by this work. However, we note that Assumption 3.1 is very natural and all rules in the form of

$$\wedge_{i=1}^k \mathtt{r}_i(A_i, B_i) \implies \mathtt{r}_{k+1}(A_k, B_k)$$

satisfy Assumption 3.1. In fact, most commonly used logic rules such as the symmetric rules, inverse rules, subrelation rules and composition rules are covered by Assumption 3.1. For example, the rules $\mathtt{BornIn}(A, B) \implies \mathtt{LiveIn}(A, B)$ and $\mathtt{BornIn}(A, B) \wedge \mathtt{CityOf}(B, C) \implies \mathtt{Nationality}(A, C)$ all satisfy Assumption 3.1.

Based on Assumption 3.1, we define the *effective length* of a logic rule, which is a key quantity in our analysis.

Definition 1. Given a logic rule F that satisfies Assumption 3.1, we define the effective length of F as its number of negated literals. That is the cardinality of $\mathcal{I}_F^-$, i.e., $|\mathcal{I}_F^-|$.

For example, the effective length of the rule $!\mathtt{r}_1(A, B) \vee \mathtt{r}_2(B, A)$ is 1 and the effective length of the rule $!\mathtt{r}_1(A, B) \vee !\mathtt{r}_2(B, C) \vee \mathtt{r}_3(A, C)$ is 2.

Assumption 3.2. *We assume that $w_i \geq 0$ for all $i \in [m]$, where w_i is the weight of the i-th rule.*

The requirement of positive weights stems from the WalkSAT algorithm, and existing MLN engines usually handle negative weights by flipping the corresponding rules and assigning positive weights to them. However, in our case, the flipped rules may no longer satisfy Assumption 3.1, and this can further break the sparsity of violated clauses (Theorem 5.1). Consequently, we constrain the weights of all rules to be non-negative. We note that Assumption 3.2 does not hinder the practical use of MLN4KB as widely used rule mining systems such as Neural LP [54] and amie [13] all produce logic rules with only positive weights. Next we introduce the open-world assumption, which is standard in the literature of MLN [8].

Assumption 3.3 (Open-world assumption). *The assignments to all unobserved facts are undecided boolean variables.*

The open-world assumption requires us to make predictions for all unobserved facts $\mathcal{H}$ during inference. Contrary to the open-world assumption, close-world assumption assumes that some relations are closed, namely the unobserved facts associated with these relations are treated as false, i.e., $\mathtt{r}(h, t) = 0$ $\forall (h, \mathtt{r}, t) \in \mathcal{H}$. Closed relations already made assignments for all associated unobserved facts and thus do not need further prediction for them. The close-world assumption can significantly reduce the number of undecided variables but with an obvious cost of losing expressiveness. Most existing MLN inference software such as Alchemy, Tuffy, DeepDive, and PSL are essentially designed under the close-world assumption and are often slow under the open-world assumption due to a large number of undecided variables. Different from the classic MLN software, MLN4KB is designed to handle the more challenging open-world assumption; inference under the close-world assumption is also included as a trivial extension.

For the ease of reference, we summarize the notations in Table 1.

4 THE MAP INFERENCE OF MLN

Given the assignment of observed facts $\mathbf{O}$, the maximum a posteriori (MAP) inference for the unobserved facts is to solve the following optimization problem:

$$\mathbf{H} = \underset{\mathbf{H} \in \{0,1\}^{|\mathcal{H}|}}{\arg\max} \sum_{i=1}^m w_i l_i(\mathbf{O}, \mathbf{H}) = \underset{\mathbf{H} \in \{0,1\}^{|\mathcal{H}|}}{\arg\min} \sum_{i=1}^m w_i n_i(\mathbf{O}, \mathbf{H}). \tag{4}$$

The MAP inference (4) is essentially a binary programming problem. Specifically, it is well-established that (4) can be formulated as a weighted maximum satisfiability (weighted MAX-SAT) problem [8], which is a classic NP-hard problem. As a result, classic MLN

Notation	Description
m	Number of rules
$\mathcal{E}$	The set of entities
$\mathcal{R}$	The set of relations
$\{F_i, w_i\}_{i=1}^m$	The set of logic rules and their weights
$O, \mathbf{O}$	Observed facts and their assignments
$\mathcal{H}, \mathbf{H}$	Unobserved facts and their assignments
$n_i(\mathbf{O}, \mathbf{H})$	The number of times that the i-th rule is violated given the assignment $\mathbf{O} \cup \mathbf{H}$
$K_{\mathrm{r}}(\mathbf{O}, \mathbf{H})$	The number of positive facts associated with relation r under the assignment $\mathbf{O} \cup \mathbf{H}$
$A, B, \ldots$	Variables in rules
$a, b, \ldots$	Grounding of variables (entities)

Table 1: Notations

inference software such as Alchemy and Tuffy are both built on local-search-based MAX-SAT solver such as the WalkSAT algorithm [17]. The MAP inference of MLN poses the following challenges:
(a) Problem (4) has $|\mathcal{H}|$ binary variables, which can be in the order of $\Theta(|\mathcal{E}|^2|\mathcal{R}|)$ under the open-world assumption. Consequently, even storing such a large number of variables is prohibitive for large-scale knowledge bases.
(b) Formulating (4) as a MAX-SAT problem requires grounding (or instantiating) first-order logic rules into clauses. It is known that the grounding of rules is #P-complete in the length of the logic rules [8, Proposition 4.1]. Therefore the number of grounded clauses is usually much larger than the number of binary variables. Indeed, the grounding phase is usually both the computational and storage bottleneck of existing MLN inference software and prevents them from scaling up to large-scale knowledge bases.
(c) Solving the weighted MAX-SAT problem is in general NP-hard, thus it is not realistic to target for an exact solution of (4) on current computing devices. Fortunately, existing local-search-based algorithms, such as the MaxWalkSAT algorithm, are able to give us an approximate solution of a medium-size MAX-SAT in a short period of time. We consider **(a)** and **(b)** as the main computational bottlenecks of existing MLN engines.

Motivated by the above challenges of MLN inference, we propose MLN4KB, an efficient MLN inference engine that can circumvent forming most variables and grounded clauses by exploiting the structure of logic rules and the sparsity of knowledge bases.

5 MLN4KB

In this section, we describe the main ideas behind the design of MLN4KB. Before proceeding to the details of MLN4KB, we first review the classic WalkSAT algorithm, which is a simple and effective stochastic search method to solve the MAX-SAT problem. The detailed WalkSAT algorithm under the scenario of MLN inference is shown in Algorithm 1. The major issue of Algorithm 1 is that the number of grounded clauses, i.e., $|C|$ can be enormous for large-scale knowledge bases. Classic MLN inference engines such as Alchemy, Tuffy and PSL usually get stuck at the construction of C and cannot even reach the WalkSAT algorithm (or other optimization subroutines).

Algorithm 1 The WalkSAT algorithm for MLN inference

```
1:  Input: A set of grounded clauses $C$, a set of unobserved facts $\mathcal{H}$, the max iteration number maxIter and a threshold $\tau$.
2:  Initialize an assignment $\mathbf{H}^{(0)}$ for unobserved facts;
3:  Set bestCost = $+\infty$, $\mathbf{H}^* = \mathbf{H}^{(0)}$;
4:  for $t \leftarrow 0, 1, \ldots,$ maxIter do
5:      Uniformly random sample a clause that is violated $c \in C$;
6:      Skip this iteration if all facts associated with $c$ are observed;
7:      Uniformly random generate a scalar $\sigma \in [0, 1]$;
8:      if $\sigma \geq \tau$ then
9:          Randomly flip the assignment of an unobserved fact that appears in $c$;
10:     else
11:         Greedily flip the assignment of an unobserved fact in $c$ such that can mostly decrease the cost;
12:     end if
13:     Update $\mathbf{H}^{(t)} \rightarrow \mathbf{H}^{(t+1)}$ and compute the cost curCost;
14:     if curCost < bestCost then
15:         bestCost = curCost, $\mathbf{H}^* = \mathbf{H}^{(t+1)}$;
16:     end if
17: end for
18: Output: the best assignment found $\mathbf{H}^*$.
```

The key insight of MLN4KB is that the construction and storage of all grounded clauses are not really necessary for the implementation of WalkSAT. WalkSAT only requires

- a subroutine to randomly generate a violated clause;
- a subroutine to select a fact in the chosen violated clause;
- a subroutine to efficiently compute the cost.

Instead of storing all grounded clauses (or some grounded clauses via lazy grounding), we can only maintain the clauses that are violated in memory and keep updating the violated clauses during the random flipping procedure. By exploiting the sparsity of knowledge bases and the structure of first-order logic rules, we can show that the number of violated clauses is far less than the total number of grounded clauses. The following theorem makes this precise.

Theorem 5.1 (Sparse violation). *Suppose that Assumption 3.1 and Assumption 3.3 hold. Follow the notations in Table 1 and denote the effective length of the i-th rule as L_i. Given an assignment of all facts $\mathbf{O} \cup \mathbf{H}$, it holds that*

$$\sum_{i=1}^{m} n_i(\mathbf{O}, \mathbf{H}) \;\leq\; \sum_{i=1}^{m} \left(\max_{\mathrm{r} \in F_i} K_{\mathrm{r}}(\mathbf{O}, \mathbf{H}) \right)^{L_i}.$$

Note that the total number of clauses generated with $\{F_i\}_{i=1}^m$ can be in the order of $\Theta(m(|\mathcal{E}|^{2L_i})$. Therefore Theorem 5.1 suggests that the total number of violated clauses is far less than the total number of clauses if most relations observe sparse positive facts with the assignment $\mathbf{O} \cup \mathbf{H}$, i.e.,

$$\max_{r \in \mathcal{R}} K_{\mathrm{r}}(\mathbf{O}, \mathbf{H}) \ll |\mathcal{E}|^2. \tag{5}$$

Consider setting all unobserved facts to false as our initialization, i.e., $\mathbf{H}^{(0)} = \{\mathrm{r}(h, t) = 0 \mid (h, \mathrm{r}, t) \in \mathcal{H}\}$, then the condition (5) holds for most real-world datasets as the number of observed facts is usually far less than the total number of facts $|\mathcal{R}||\mathcal{E}|^2$. Moreover, if the WalkSAT algorithm can produce an almost decreasing cost,

Algorithm 2 The MAP inference of MLN4KB

```
1: Input: A set of rules $\mathcal{F} = \{F_i\}_{i=1}^m$, a set of observed assignment $\mathbf{O}$, a set of unobserved facts $\mathcal{H}$, the max iteration number maxIter, a threshold $\tau$ and a sparsity parameter $K > 0$.
2: Initialize the assignment for the unobserved facts $\mathbf{H}^{(0)} = \{\mathsf{r}(h,t) = 0 \mid (h,\mathsf{r},t) \in \mathcal{H}\}$;
3: Construct all clauses that are violated violatedClauses;
4: Set bestCost $= +\infty$, $\mathbf{H}^* = \mathbf{H}^{(0)}$;
5: for $t \leftarrow 0, 1, \ldots,$ maxIter do
6:     Randomly sample a clause $c$ from violatedClauses;
7:     Skip this iteration if all facts associated with $c$ are observed;
8:     Uniformly random generate a scalar $\sigma \in [0,1]$;
9:     if $\sigma \geq \tau$ then
10:        Randomly flip the assignment of an unobserved fact $(h,\mathsf{r},t)$ that appears in $c$;
11:    else
12:        Greedily flip the assignment of an unobserved fact $(h,\mathsf{r},t)$ in $c$ such that can mostly decrease the cost;
13:    end if
14:    (optional) Flip back the fact $(h,\mathsf{r},t)$ and skip this iteration if the constraint $K_{\mathsf{r}}(\mathbf{O},\mathbf{H}) \leq K$ is violated;
15:    Loop over clauses in violatedClauses that are related to $(h,\mathsf{r},t)$, remove the clause if it is satisfied;
16:    Construct violated clauses that involves $(h,\mathsf{r},t)$ and insert them into violatedClauses;
17:    Update $\mathbf{H}^{(t)} \rightarrow \mathbf{H}^{(t+1)}$ and compute the cost curCost;
18:    if curCost < bestCost then
19:        bestCost = curCost, $\mathbf{H}^* = \mathbf{H}^{(t+1)}$;
20:    end if
21: end for
22: Output: the best assignment found $\mathbf{H}^*$.
```

which is usually the case in practice, then we can further bound the number of violated clauses among all iterations of Algorithm 2. Formally, we have the following corollary.

Corollary 5.2. *Follow the notations in Table 1 and denote the effective length of the i-th rule as L_i. Let $\{\mathbf{H}^{(t)}\}_{t=0}^T$ be the iterates generated from Algorithm 2, where T is the total number of iterations. Denote $f(\mathbf{O},\mathbf{H}^{(t)}) = \sum_{i=1}^m w_i n_i(\mathbf{O},\mathbf{H}^{(t)})$ as the cost at the t-th iteration. If $f(\mathbf{O},\mathbf{H}^{(t)}) \leq Cf(\mathbf{O},\mathbf{H}^{(0)})\ \forall t \in [T]$ for some $C > 0$, then*

$$\sum_{i=1}^m n_i(\mathbf{O},\mathbf{H}^{(t)}) \ \leq\ Cw_{\min}^{-1} \sum_{i=1}^m \left(\max_{\mathsf{r}\in\mathcal{R}} \sum_{(h,\mathsf{r},t)\in O} \mathsf{r}(h,t) \right)^{L_i}$$

for any $t \in [T]$, where $w_{\min} = \min_{i\in[m]} w_i$.

In Section 7, we empirically verify that Algorithm 2 can yield bounded cost and thus maintain a reasonable amount of violated clauses during the inference phase. Another possible strategy to control the memory usage of Algorithm 2 is to directly impose additional cardinality constraints on the number of positive facts, for example $K_{\mathsf{r}}(\mathbf{O},\mathbf{H}) \leq K$ for some $K > 0$, then the inequality stated in Theorem 5.1 reduces to

$$\sum_{i=1}^m n_i(\mathbf{O},\mathbf{H}) \ \leq\ \sum_{i=1}^m K^{L_i}.$$

It is easy to modify Algorithm 2 to incoporate the above constraint.

Motivated by Theorem 5.1, MLN4KB does not ground out all clauses but stores only the clauses that are violated under the current assignment in memory, we name the variable that stores violated clauses as `violatedClauses`. In order to support efficient sampling, insertion, deletion, and look-up operations for violated clauses, MLN4KB uses a combination of vector and dictionary as the data structure for `violatedClauses`. In every iteration of the WalkSAT algorithm, MLN4KB performs the following steps:

(1) MLN4KB first uniform randomly selects a clause that is violated under the current assignment, i.e., uniform randomly sample an element from `violatedClauses`;
(2) then MLN4KB either randomly or greedily selects a fact and flip its assignment (either $0 \rightarrow 1$ or $1 \rightarrow 0$);
(3) after flipping the assignment of the selected fact, some clauses in `violatedClauses` become satisfied and some previously satisfied clauses become violated now, MLN4KB removes satisfied clauses and inserts new violated clauses to `violatedClauses` by checking all clauses that are related to the flipped fact;
(4) MLN4KB computes the cost of the new assignment and updates the solution if the new cost is better than the best record.

The detailed algorithm of the inference subroutine of MLN4KB is given in Algorithm 2. The time complexities of steps (1), (2) are obviously $O(1)$; step (3) can be realized by back-tracking depth-first-search; the cost of step (4) is decided by the number of removal and insertion from step (3). Formally, the time and space complexity of Algorithm 2 is characterized by the following theorem.

Theorem 5.3 (Space and time complexities). *Assume that the constraint $K_{\mathsf{r}}(\mathbf{O},\mathbf{H}) \leq K$ for some $K > 0$ is imposed $\forall\ \mathsf{r} \in \mathcal{R}$. Then the worst-case space complexity and per iteration time complexity of Algorithm 2 is $O(K|\mathcal{R}| + \sum_{i=1}^m K^{L_i})$ and $O(\sum_{i=1}^m K^{L_i})$ respectively.*

According to Theorem 5.3, when K and `maxIter` are small, the time and space complexities of the MAP inference of MLN4KB can be far less than the grounding phase in classic MLN engines. The efficiency of MLN4KB on large-scale knowledge bases is demonstrated in Section 7.

Remark 5.1 (Difference to lazy inference). MLN4KB stores only violated clauses in memory and circumvents the efficiency issue of the grounding phase required by existing MLN engines. This design is different from the *lazy inference* or *lazy grounding* technique [8, 36, 43] used in existing MLN engines such as Alchemy, Tuffy and DeepDive. Lazy inference also exploits the sparseness of positive facts; it stores clauses that can be violated by flipping some *active* fact (active facts are facts that have been flipped during execution). Lazy inference can also reduce the number of stored clauses, but its storage is still far more than the number of violated clauses. This difference is also demonstrated in Section 7 as MLN4KB is more time and space efficient than existing MLN engines equipped with lazy inference.

5.1 Weight learning

To support the complete functionality of an MLN solver, we also include a weight learning module to MLN4KB. Various learning algorithms of MLN have been proposed in the literature. For example, the generative learning [8, 15], discriminative learning [16], learning based on pseudo-likelihood [38], gradient boosting [18] and random walk [21], etc. Among these approaches, we find that discriminative learning and optimizing the pseudo-likelihood best fit the design of MLN4KB. Specifically, the discriminative learning approach repeatedly calls the MAP inference subroutine to perform stochastic gradient step [16], and the learning based on pseudo-likelihood approach aims at maximizing the pseudo-log-likelihood

$$\max_{w} \sum_{(h,\mathsf{r},t)\in \mathrm{O}\cup \mathrm{H}} \log \Pr\left[\mathsf{r}(h,t) = \mathbf{1}_{\{(h,\mathsf{r},t)\in \mathrm{O}\}} \,\middle|\, MB(h,\mathsf{r},t)\right],$$

where $\mathbf{1}_{\{\cdot\}}$ is the indicator function, i.e., $\mathbf{1}_{true} = 1$ and $\mathbf{1}_{false} = 0$, and $MB(h,\mathsf{r},t)$ denotes the Markov blanket of the fact (h,r,t), and the probability distribution is defined in eq. (3).

For large-scale knowledge bases, optimizing the pseudo-likelihood is usually more efficient than the discriminative learning approach because the latter usually requires invoking the MAP inference at least hundreds of times, which can be prohibitively expensive in practice. Therefore we adopt the pseudo-likelihood maximization as the default learning algorithm of MLN4KB. Classic MLN learning algorithm optimizes the pseudo-likelihood function via the scaled stochastic gradient descent or the L-BFGS optimizer [8, §4.1]. Thanks to the recent progress in stochastic optimization made by pioneering researchers in the past decade, we include more advanced optimizers, including the Adagrad [11] and Adam [20] optimizer into MLN4KB. In addition to the classic weight learning, we also include a positive and unlabeled (PU) weight learning subroutine given that many real-world KB datasets are positive and unlabeled. The material presented in this subsection is mostly based on existing algorithms, and we do not claim much technical novelty for the learning module of MLN4KB.

6 IMPLEMENTATION

We implement MLN4KB as a Julia package called MLN4KB.jl. Julia [2] is an emerging programming language for high-performance numerical computation and has almost all the functionalities needed for a high-performance MLN engine. Moreover, a Julia project is easy to develop and its code is reader-friendly; this allows potential interested researchers and practitioners to understand and conduct future research based on MLN4KB.jl conveniently.

7 EXPERIMENTS

We conduct experiments to evaluate the efficiency and effectiveness of MLN4KB.jl. We use the following datasets and adopt the train/test splitting from TransE [6] and Neural LP [54].

- **Kinship**: Kinship is a synthetic dataset that contains kinship relationships such as `father`,`mother`,`husband`,`wife`, etc. To compare the scalability of different algorithms, we generate a series of Kinship datasets with different number of entities ranges in $\{10^2, 5\times10^2, 10^3, 5\times10^3, 10^4, 10^5\}$. Following the standard evaluation setup in the literature [54], we use the observed kinship relations to predict entities' gender.
- **UMLS**: Unified medical language system (UMLS) [4] is a benchmark dataset for statistical relation learning in the field of biomedicine. It contains 6,529 facts (5,896 training and 633 testing facts), 135 entities and 46 relations.
- **WN-18**: WN-18 is generated from the WordNet [19] and has been widely used as a benchmark dataset for knowledge graph completion. WN-18 has 151,442 facts (14,6442 training and 5,000 testing facts), 40,943 entities and 18 relations.
- **YAGO3-10**: YAGO3-10 is a subset of the YAGO knowledge base [45] and is a standard benchmark dataset for knowledge graph completion. YAGO3-10 contains 1,084,040 facts (1,079,040 training and 5,000 testing facts), 123,143 entities and 37 relations.

We include Tuffy[1], DeepDive[2] and PSL[3] as the baseline methods into comparison. Note that these software are well-engineered MLN inference and learning engines, therefore it is fair to compare with their runtime. For the Kinship datasets, we use the standard kinship rules as the logic rules and set the weights of all rules to be 1.0. For UMLS, WN-18 and YAGO3-10, we use the amie [13] rule mining system to extract logic rules and use the weight learning module of MLN4KB.jl to learn the weights of the extracted logic rules. We set the threshold parameter τ in Algorithm 2 to be 0.1 as the default value if not otherwise specified. All experiments are conducted on a machine with 32 Intel CPUs and 64GB memory.

Outline of experiments: We compare the inference time of MLN4KB.jl against existing competitive MLN solvers in Section 7.1 and demonstrate the importance of the open-world assumption in Section 7.2. In Section 7.3, we evalute MLN4KB.jl and other baseline MLN solvers on real-world knowledge base datasets. We evaluate the memory usage and hyper-parameter sensitivity of MLN4KB.jl in Section 7.4 and Section 7.5.

7.1 Comparing inference time on Kinship

We compare the inference time of different MLN engines by using the Kinship datasets with a different number of entities ranging from 10^2 to 10^5. The experimental results in both open-world and close-world settings are shown in Table 2. Note that the kinship datasets used in this experiment do not contain any noise, thus all methods can achieve nearly 100% testing accuracy, thus we omit the prediction accuracy in Table 2. From Table 2, we observe that

- in the open-world setting, i.e., the MLN engine is required to infer the status of the missing facts for all relations simultaneously, MLN4KB.jl is orders of magnitudes faster (at least 10^3 times faster) than all other baseline MLN engines for all Kinship datasets. When the number of entities is greater than $10,000$, all baseline methods either run out of time (more than 8 hours) or run out of space, while MLN4KB.jl can solve all cases within one minute.
- in the less challenging close-world setting, i.e., the MLN engine only needs to infer the gender information of each entity, both Tuffy, DeepDive and MLN4KB.jl are able to handle the Kinship dataset with 10^5 entities. Both Tuffy and MLN4KB.jl can solve all cases within 20 seconds. MLN4KB.jl is still the fastest among all MLN engines for all cases.

The experimental results in this section strongly support the efficiency of MLN4KB.jl, especially for large-scale knowledge bases in the challenging open-world setting.

[1]http://i.stanford.edu/hazy/tuffy/
[2]http://deepdive.stanford.edu/
[3]https://psl.linqs.org/

		Number of entities					
Settings	Algorithms	100	500	1,000	5,000	10,000	100,000
open-world	Tuffy	2s	30s	~1min	~4.5hr	NA	NA
	DeepDive	~4min	~4hr	NA	NA	NA	NA
	PSL	20s	~14min	NA	NA	NA	NA
	MLN4KB.jl	**0.001s**	**0.005s**	**0.007s**	**0.9s**	**2.1s**	**33.5s**
close-world	Tuffy	0.9s	1.0s	1.1s	1.9s	2.6s	18.3s
	DeepDive	1min	1min	1min	1min	1min	1min10s
	PSL	20s	~14min	NA	NA	NA	NA
	MLN4KB.jl	**0.001s**	**0.009s**	**0.01s**	**0.04s**	**0.18s**	**8.9s**

Table 2: Comparison of inference time with the Kinship dataset under the open-world and close-world settings. NA means that the method either run out of memory (or disk) or cannot finish inference within 8 hours. For each setting, we highlight the shortest inference time.

	Missing percentage					
Settings	0%	10%	20%	30%	40%	50%
open-world	**100**	**99.5**	**98.08**	**96.48**	**92.18**	**86.36**
close-world	**100**	98.8	95.68	91.42	83.94	75.16

Table 3: The prediction accuracy of MLN4KB.jl for Kinship dataset (5000 entities) with different levels of noise.

7.2 Open-world assumption matters

In this part, we design an experiment to illustrate the importance of the open-world assumption. Again, we use the Kinship datasets for evaluation. Instead of using the Kinship datasets with complete relation information, we uniform randomly drop some observed facts with a certain probability. We try the dropping rates in $\{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$. The prediction accuracy of MLN4KB.jl for Kinship (5,000 entities) with various dropping rates under the open-world and close-world settings is given in Table 3. As shown in Table 3, we can observe that MLN4KB.jl in the open-world setting consistently outperforms the its predictions in the close-world setting. The performance gap between open-world and close-world settings becomes more obvious as the dropping rate goes up. The observations in this section should not be surprising. MLN with the close-world assumption cannot fill in the missing facts for closed relations, and the missing facts of closed relations will further affect the predictions for the open relations. Our experiment demonstrates the superiority of the open-world setting, and therefore further supports the significance of MLN4KB.jl as MLN4KB.jl is able to handle large knowledge bases in the open-world setting.

7.3 Evaluation on real-world KB datasets

We evaluate different MLN engines on the *link prediction* task with real-world knowledge bases. Given a fact (or query) (h, r, t) from the testing set, the evaluation of the link prediction task is based on the rank of t among all candidate tail entities associated with the head h and relation r. Note that the MAP inference of MLN assigns either 0 or 1 to each fact, and it tends to produce a large number of ties among the facts assigned with the same value. In order to break the ties and calculate a meaningful rank of the given tail entity t, we sort the tail entities with the same score by their marginal contribution to the total cost, i.e., how much the cost decreases by flipping the fact from 0 to 1.

		Datasets		
Algorithms	Metrics	UMLS	WN-18	YAGO3-10
Tuffy	–	NA	NA	NA
DeepDive	–	NA	NA	NA
PSL	–	NA	NA	NA
MLN4KB.jl	MRR	48.32	69.40	41.19
	Hit@5	82.31	95.18	60.38
	Hit@10	91.15	96.62	62.54
	Inference time	~5min	~30min	~10hr

Table 4: The testing accuracy of different methods on real-world knowledge base datasets.

Evaluation metrics. For the link prediction task, we follow the literature and adopt the filtered evaluation setting [6], that is all facts appeared in the training and testing (except the one of being tested) are removed when evaluating the rank of a given testing fact. We use Hit@k and the mean reciprocal rank (MRR) as our evaluation metrics, where Hit@k measures the portion of test facts that are ranked in top k against other candidate facts and MRR is the averaged inverse of the rank of testing facts. We try $k \in \{5, 10\}$ in our experiments.

Logic rules. For each knowledge bases, we first use the rule mining system amie [13] to generate a set of candidate first-order logic rule. Then we call the learning module of MLN4KB.jl to calculate the weights of the extracted rules; rules weighted with 0 after learning are discarded during inference. After rule extraction, learning and pruning, we obtain 1,038 rules for UMLS, 98 rules for WN-18 and 206 rules for YAGO3-10.

Results. The testing accuracy of MLN4KB.jl on UMLS, WN-18, YAGO3-10 and their corresponding inference time are shown in Table 4. From Table 4, we observe that Tuffy, DeepDive and PSL can not solve either problem in the open-world setting while MLN4KB.jl can finish the inference within a reasonable amount of time. For WN-18 and YAGO4-10, the Hit@10 obtained by MLN4KB.jl are comparable to the state-of-the-art results achieved by the heavily-tuned embedding-based methods [7, 49]. To our knowledge, this is the first time that pure MLN method can scale to WN-18 and YAGO3-10 in the open-world setting without any approximation tricks. We note that we simply adopt the default settings of MLN4KB.jl

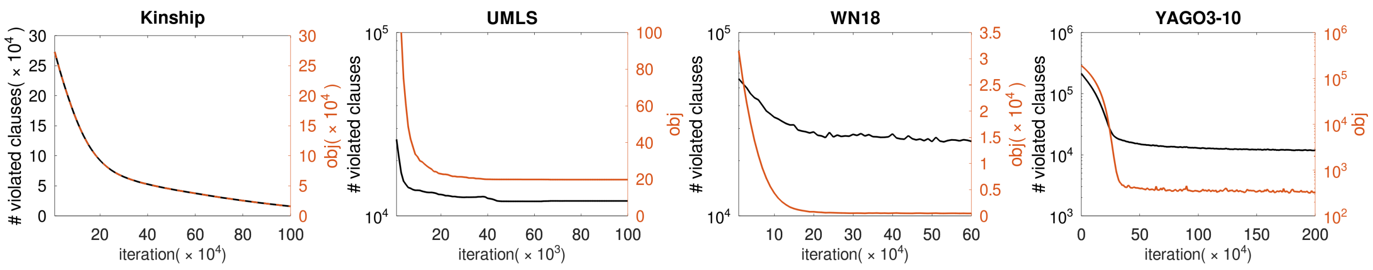

Figure 2: The evolution of the cost (objective value) along with the number of violated clauses during inference. The black curve represents the evolution of the number of violated clauses and the red curve standards for the evolution of the cost. The two curves of Kinship coincide because we set the weights of all rules to be 1 for the Kinship datasets.

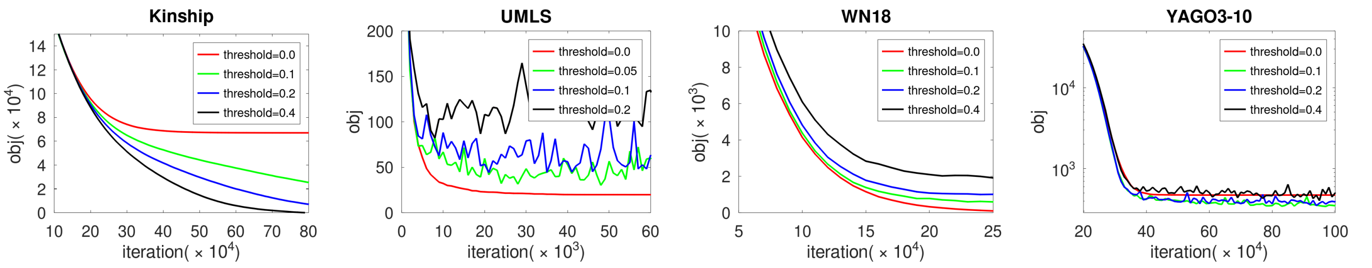

Figure 3: The evolution of the cost (objective value) with different threshold parameters.

Kinship (10^5)	UMLS	WN-18	YAGO3-10
167MB	318MB	196MB	1.08GB

Table 5: The RAM usage of MLN4KB.jl.

(setting the threshold τ to 1 and the number of negative sample to 1 during the weight learning) for the experiments in this section, and more careful rule mining and hyper-parameter tunning may further improve the testing accuracy.

We also attempted to apply MLN4KB.jl to the widely used FB15k datasets [5]. Unfortunately, FB15k contains more than 1,000 relations and amie extracted more than 4×10^4 rules, MLN4KB.jl can not finish learning and inference within a day. We left further accelerating MLN4KB.jl via parallelization and warm-up initialization as our future work.

7.4 The memory usage of MLN4KB.jl

We examine the memory usage of MLN4KB.jl without any hard cardinality constraints on the sparsity of positive facts. The main memory usage of MLN4KB.jl is spent on storing the knowledge base and the set of violated clauses. For the storage of knowledge base, the memory usage is mainly controlled by the number of observed facts, which is usually affordable when the observation in knowledge base is sparse. For the set of violated clauses, we plot the evolutions of the total cost and the number of violated clauses during inference on different real-world knowledge bases in Figure 2. From Figure 2, we observe that both the cost and the number of violated clauses tend to keep decreasing during the inference, which is consistent with our theoretical analysis in Section 5. The detailed peak memory usages of MLN4KB.jl on different knowledge bases are shown in Table 5; the memory usage of MLN4KB.jl does not exceed 1.1GB for all testing knowledge bases. We conclude that MLN4KB.jl is memory-efficient on real-world knowledge bases.

7.5 Sensitivity to the threshold parameter

We test the effects of the threshold parameter τ in Algorithm 2. The evolutions of the total cost with different thresholds on various knowledge bases are shown in Figure 3. We observe that the threshold parameter has a significant impact on the convergence of the cost, and different knowledge bases favour different thresholds. For example, setting $\tau = 0$ (which is equivalent to greedy search) yields the best convergence for the UMLS dataset, while setting $\tau = 0.1$ fits best to the YAGO3-10 dataset.

To balance the trade-off between exploration and exploitation in the local search algorithm, we set the default threshold τ to be 0.1 for MLN4KB.jl. In practice, we encourage practitioners to try different thresholds and select the best for their applications.

7.6 Conclusion and future work

In this work, we develop an efficient MLN engine by designing smart algorithm to leverage the structure of a certain class of logic rules and the sparsity of knowledge bases. We implement the proposed MLN engine as a Julia package called MLN4KB.jl. Experiments on both synthetic and real-world knowledge bases strongly support the effectiveness of MLN4KB.jl.

Future directions remain. Firstly, the current implementation of MLN4KB.jl still does not scale to knowledge bases with a large number of relations and rules. Further acceleration techniques such as parallelization, efficient greedy search, and more careful memory management are needed for MLN4KB.jl. We left adding these acceleration techniques to MLN4KB.jl as our future work. Secondly, MLN4KB.jl has some limitations. In particular, it does not support negative weights and marginal inference. It is worth to explore if it is possible to realize these functionalities under the framework of MLN4KB.jl. Lastly, it would be interesting to explore whether MLN4KB.jl can break the state-of-the-art testing accuracy on benchmark knowledge bases by a more careful mining of rules.

REFERENCES

[1] Bach, S. H., Broecheler, M., Huang, B., and Getoor, L. (2017). Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research*, 18:109:1–109:67.

[2] Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.

[3] Biba, M. (2009). Integrating logic and probability: Algorithmic improvements in markov logic networks. *PhD Thesis. University of Bari.*

[4] Bodenreider, O. (2004). *Nucleic Acids Res.*, 32:267–270.

[5] Bollacker, K. D., Evans, C., Paritosh, P. K., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, (SIGMOD)*, pages 1247–1250. ACM.

[6] Bordes, A., Usunier, N., García-Durán, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 2787–2795.

[7] Dettmers, T., Minervini, P., Stenetorp, P., and Riedel, S. (2018). Convolutional 2d knowledge graph embeddings. In *The AAAI Conference on Artificial Intelligence*, pages 1811–1818. AAAI Press.

[8] Domingos, P. and Lowd, D. (2009a). *Markov Logic: An Interface Layer for Artificial Intelligence.* Morgan & claypool publishers.

[9] Domingos, P. M. and Lowd, D. (2009b). Markov logic: Theory, algorithms and applications. *PhD Thesis. University of Washington, Seattle.*

[10] Domingos, P. M. and Lowd, D. (2019). Unifying logical and statistical AI with markov logic. *Commun. ACM*, 62(7):74–83.

[11] Duchi, J. C., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

[12] Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. (1999). Learning probabilistic relational models. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1300–1309.

[13] Galárraga, L. A., Teflioudi, C., Hose, K., and Suchanek, F. M. (2013). AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *The International World Wide Web Conference (WWW)*, pages 413–422.

[14] Genesereth, M. R. and Nilsson, N. J. (1988). *Logical foundations of artificial intelligence.* Morgan Kaufmann.

[15] Haaren, J. V., den Broeck, G. V., Meert, W., and Davis, J. (2016). Lifted generative learning of markov logic networks. *Machine Learning*, 103(1):27–55.

[16] Huynh, T. N. and Mooney, R. J. (2008). Discriminative structure and parameter learning for markov logic networks. In *Proceedings of the International Conference on Machine Learning, ICML*, volume 307, pages 416–423.

[17] Kautz, H. A., Selman, B., and Jiang, Y. (1996). A general stochastic approach to solving problems with hard and soft constraints. In *Satisfiability Problem: Theory and Applications*, volume 35, pages 573–585.

[18] Khot, T., Natarajan, S., Kersting, K., and Shavlik, J. W. (2015). Gradient-based boosting for statistical relational learning: the markov logic network and missing data cases. *Machine Learning*, 100(1):75–100.

[19] Kilgarriff, A. and Fellbaum, C. (2000). Wordnet, an electronic lexical database.

[20] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[21] Kok, S. and Domingos, P. M. (2010). Learning markov logic networks using structural motifs. In *Proceedings the International Conference on Machine Learning (ICML)*, pages 551–558.

[22] Kok, S., Sumner, M., Richardson, M., Singla, P., Poon, H., Lowd, D., and Domingos, P. (2000). The alchemy system for statistical relational ai. *Technical Report. Department of Computer Science and Engineering, University of Washington, Seattle, WA.*

[23] Lifschitz, V. (1990). *Formalizing Common Sense: Papers by John McCarthy.* Norwood, New Jersey: Ablex Publishing Corporation.

[24] McCarthy, J. (1959). Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes.*

[25] McCarthy, J. (1979). First order theories of individual concepts and propositions. In *Machine Intelligence.*

[26] McCarthy, J. and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence.*

[27] Mihalkova, L. and Mooney, R. J. (2007). Bottom-up learning of markov logic network structure. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 227, pages 625–632. ACM.

[28] Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, 8(4):295—-318.

[29] Muggleton, S. (1996). Stochastic logic program. *Advances in Inductive Logic Programming*, 7:254–264.

[30] Nickel, M., Rosasco, L., and Poggio, T. A. (2016). Holographic embeddings of knowledge graphs. In *The AAAI Conference on Artificial Intelligence*, pages 1955–1961. AAAI Press.

[31] Niepert, M., Meilicke, C., and Stuckenschmidt, H. (2010). A probabilistic-logical framework for ontology matching. In *The AAAI Conference on Artificial Intelligence.* AAAI Press.

[32] Niu, F., Ré, C., Doan, A., and Shavlik, J. W. (2011). Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS. *The International Journal on Very Large Data Bases (VLDB)*, 4(6):373–384.

[33] Poon, H. and Domingos, P. M. (2006). Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of the Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*, pages 458–463.

[34] Poon, H. and Domingos, P. M. (2007). Joint inference in information extraction. In *The AAAI Conference on Artificial Intelligence*, pages 913–918. AAAI Press.

[35] Poon, H. and Domingos, P. M. (2009). Unsupervised semantic parsing. In *Proceedings the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–10. ACL.

[36] Poon, H., Domingos, P. M., and Sumner, M. (2008). A general method for reducing the complexity of relational inference and its application to MCMC. In *The AAAI Conference on Artificial Intelligence.*

[37] Qu, M. and Tang, J. (2019). Probabilistic logic neural networks for reasoning. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 7710–7720.

[38] Richardson, M. and Domingos, P. M. (2006). Markov logic networks. *Machin. Learning*, 62(1-2):107–136.

[39] Riedel, S. (2008). Improving the accuracy and efficiency of MAP inference for markov logic. In *Proceedings of the Conference in Uncertainty in Artificial Intelligence*, pages 468–475.

[40] Sa, C. D., Ratner, A., Ré, C., Shin, J., Wang, F., Wu, S., and Zhang, C. (2016). Deepdive: Declarative knowledge base construction. *SIGMOD Rec.*, 45(1):60–67.

[41] Shavlik, J. W. and Natarajan, S. (2009). Speeding up inference in markov logic networks by preprocessing to reduce the size of the resulting grounded network. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1951–1956.

[42] Shin, J., Wu, S., Wang, F., Sa, C. D., Zhang, C., and Ré, C. (2015). Incremental knowledge base construction using deepdive. *The International Journal on Very Large Data Bases (VLDB)*, 8(11):1310–1321.

[43] Singla, P. and Domingos, P. M. (2006). Memory-efficient inference in relational domains. In *The AAAI Conference on Artificial Intelligence*, pages 488–493.

[44] Singla, P. and Domingos, P. M. (2008). Lifted first-order belief propagation. In *The AAAI Conference on Artificial Intelligence*, pages 1094–1099. AAAI Press.

[45] Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: a core of semantic knowledge. In *Proceedings of the International Conference on World Wide Web (WWW)*, pages 697–706. ACM.

[46] Sun, Z., Deng, Z., Nie, J., and Tang, J. (2019). Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations, ICLR.*

[47] Sun, Z., Zhao, Y., Wei, Z., Zhang, W., and Wang, J. (2017). Scalable learning and inference in markov logic networks. In *International Journal of Approximate Reasoning*, volume 82, pages 39–55.

[48] Thomason, R. (2020). Logic and Artificial Intelligence. In *The Stanford Encyclopedia of Philosophy.* Metaphysics Research Lab, Stanford University.

[49] Tran, H. N. and Takasu, A. (2022). MEIM: multi-partition embedding interaction beyond block term format for efficient and expressive link prediction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 2262–2269.

[50] Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., and Bouchard, G. (2016). Complex embeddings for simple link prediction. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2071–2080.

[51] Wellman, M., Breese, J. S., and Goldman, R. P. (1992). From knowledge bases to decision models. *Knowledge Engineering Review*, 7:35–53.

[52] Wielemaker, J. (2003). An overview of the SWI-Prolog programming environment. In Mesnard, F. and Serebenik, A., editors, *Proceedings of the 13th International Workshop on Logic Programming Environments*, pages 1–16. Katholieke Universiteit Leuven.

[53] Wielemaker, J. (2009). *Logic programming for knowledge-intensive interactive applications.* PhD thesis, University of Amsterdam. http://dare.uva.nl/en/record/300739.

[54] Yang, F., Yang, Z., and Cohen, W. W. (2017). Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2319–2328.

[55] Yoshikawa, K., Riedel, S., Asahara, M., and Matsumoto, Y. (2009). Jointly identifying temporal relations with markov logic. In *Proceedings of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing of the AFNLP*, pages 405–413.

[56] Zhang, Y., Chen, X., Yang, Y., Ramamurthy, A., Li, B., Qi, Y., and Song, L. (2020). Efficient probabilistic logic reasoning with graph neural networks. In *International Conference on Learning Representations, ICLR.*

APPENDIX

A MISSING PROOFS

Proof of Theorem 5.1. Consider the i-th rule F_i. Without loss of generality, we assume that F_i is of the form eq. (1). For a grounding of F_i, it is false (violated) iff all negated literals are true and all un-negated literals are false. Therefore,

$$\begin{aligned} n_i(\mathbf{O},\mathbf{H}) &= \sum_{\text{all possible entities}} \prod_{i\in\mathcal{I}^-_{F_i}} \mathsf{r}_i(h_i,t_i) \prod_{i\in\mathcal{I}^+_{F_i}} (1-\mathsf{r}_i(h_i,t_i)) \\ &\le \sum_{\text{all possible entities}} \prod_{i\in\mathcal{I}^-_{F_i}} \mathsf{r}_i(h_i,t_i) \\ &\overset{(*)}{\le} \prod_{i\in\mathcal{I}^-_{F_i}} K_{\mathsf{r}_i}(\mathbf{O},\mathbf{H}) \le \left(\max_{\mathsf{r}\in F_i} K_{\mathsf{r}}(\mathbf{O},\mathbf{H})\right)^{|\mathcal{I}^-_{F_i}|}, \end{aligned}$$

where $(*)$ uses Assumption 3.1. Summing i over $\{1,2,\ldots,m\}$ yields the desired result. □

Proof of Theorem 5.3. First, we calculate the space complexity of MLN4KB. The space complexity of MLN4KB is dominated by the storage of the knowledge base and the size of the violated clauses. With the constraint $K_{\mathsf{r}}(\mathbf{O},\mathbf{H}) \le K\ \forall \mathsf{r} \in \mathcal{R}$,

- the total number of positive facts is in the order of $O(K|\mathcal{R}|)$;
- the total number of violated clauses is $O(\sum_{i=1}^m K^{L_i})$ by Theorem 5.1.

To sum up, the space complexity of MLN4KB with the cardinality constraints is $O(K|\mathcal{R}| + \sum_{i=1}^m K^{L_i})$.

Next, we calculate the per iteration time complexity of MLN4KB. The time complexity MLN4KB is controlled by the number of violated clauses that become satisfied and the number of satisfied clauses that become violated in that iteration. The size of these clauses is bounded by $O(\sum_{i=1}^m K^{L_i})$. Therefore, the time complexity of flipping a fact in a violated clauses and updating corresponding data structures is $O(\sum_{i=1}^m K^{L_i})$. □

Proof of Corollary 5.2. For any $t \in [T]$,

$$\begin{aligned} \sum_{i=1}^m n_i(\mathbf{O},\mathbf{H}^{(t)}) &\le w_{\min}^{-1}\left(\sum_{i=1}^m w_i n_i(\mathbf{O},\mathbf{H}^{(t)})\right) \\ &= w_{\min}^{-1} f(\mathbf{O},\mathbf{H}^{(t)}) \\ &\overset{\text{(i)}}{\le} w_{\min}^{-1} C f(\mathbf{O},\mathbf{H}^{(0)}) \\ &\overset{\text{(ii)}}{\le} C w_{\min}^{-1} \sum_{i=1}^m w_i \left(\max_{\mathsf{r}\in\mathcal{R}} K_{\mathsf{r}}(\mathbf{O},\mathbf{H}^{(0)})\right)^{L_i} \\ &\overset{\text{(iii)}}{\le} C w_{\min}^{-1} \sum_{i=1}^m w_i \left(\max_{\mathsf{r}\in\mathcal{R}} \sum_{(h,\mathsf{r},t)\in O} \mathsf{r}(h,t)\right)^{L_i}, \end{aligned}$$

where (i) is by the assumption $f(\mathbf{O},\mathbf{H}^{(t)}) \le Cf(\mathbf{O},\mathbf{H}^{(0)})$, (ii) is a direct consequence of Theorem 5.1, (iii) is by the definition of K_{r} and $\mathbf{H}^{(0)}$. □