Sya: Enabling Spatial Awareness inside Probabilistic Knowledge Base Construction

Ibrahim Sabek
Department of Computer Science and Engineering
University of Minnesota, Minnesota, USA
sabek@cs.umn.edu

Mohamed F. Mokbel*
Qatar Computing Research Institute
Hamad bin Khalifa University, Doha, Qatar
mmokbel@hbku.edu.qa

***Abstract*—This paper presents *Sya*; the first spatial probabilistic knowledge base construction system, based on Markov Logic Networks (MLN). *Sya* injects the awareness of spatial relationships inside the MLN grounding and inference phases, which are the pillars of the knowledge base construction process, and hence results in a better knowledge base output. In particular, *Sya* generates a probabilistic model that captures both *logical* and *spatial* correlations among knowledge base relations. *Sya* provides a simple spatial high-level language, a spatial variation of factor graph, a spatial rules-query translator, and a spatially-equipped statistical inference technique to infer the factual scores of relations. In addition, *Sya* provides an optimization that ensures scalable grounding and inference for large-scale knowledge bases. Experimental evidence, based on building two real knowledge bases with spatial nature, shows that *Sya* can achieve 70% higher F1-score on average over the state-of-the-art DeepDive system, while achieving at least 20% reduction in the execution times.**

I. Introduction

Knowledge base construction has been an active area of research over the last two decades with several system prototypes coming from academia (e.g., [4], [6]) and industry (e.g., [7], [12], [26]), along with many important applications, e.g., web search [18], digital libraries [11], and health care [8]. The goal of knowledge base construction is to extract *factual* structured data (i.e., knowledge base) from unstructured data sources, e.g., Wikipedia, semantic web, and business logs. Examples of such facts include "Alice is a spouse of Bob" or "John has Ebola". Most recently, the idea of *probabilistic* (instead of *factual*) knowledge bases has been proposed, where each extracted *relation* is associated with a probability of how the system is confident that this relation is factual (e.g., see [9], [10], [25], [36]). An example of such probabilistic relations is "Alice is a spouse of Bob with 80% probability".

Recently, Markov Logic Networks (MLN) [34] have been a standard tool for building probabilistic knowledge base construction systems. Examples of such systems include DeepDive [36], ProbKB [9], and Archimedes [10]. In these MLN-based systems, users express the knowledge base construction logic using a set of first-order logic rules [16]. Then, such rules are processed on two steps: 1) *grounding*, which evaluates the rules to construct a ground factor graph [43] that encodes the probability distribution of all extracted knowledge base relations; and 2) *inference*, which estimates the marginal distribution (i.e., factual score) for each relation. Unfortunately, current MLN-based knowledge base construction systems do not fully utilize or acknowledge the importance of the spatial information associated with various entities in extracted relations. This immediately results in knowledge base relations with less accurate factual scores. To better illustrate this issue, we provide a real-example from epidemiology.

County	Sanitation Level	Ebola Infection Rate
Montserrado	0.6	1
Margibi	0.6	?
Bong	0.6	?
Gbarpolu	0.6	?

(a) Input to Knowledge Base System

County	Infection Rate Range	DeepDive	DeepDive + Spatial	Sya
Montserrado	[0.6, 1]	1	1	1
Margibi	[0.6, 1]	0.54	0.51	0.76
Bong	[0.4, 0.6[	0.52	0.45	0.53
Gbarpolu	[0.2, 0.4[	0.63	0.06	0.22

(b) Output from Knowledge Base System

(c) Liberia Counties Map

Fig. 1. Factual Scores of EbolaKB Using DeepDive and *Sya*.

Example. We used DeepDive [36], a popular MLN-based knowledge base construction system, to build a knowledge base about Ebola infected counties in Liberia. First, we did feed DeepDive system with data about sanitation levels [30] in various counties in Liberia, namely EbolaKB. Figure 1(a) shows a table of such information for four counties in Liberia, namely, Montserrado, Margibi, Bong, and Gbarpolu. One of these counties, Montserrado, was declared by United Nations to have a high infection rate, hence marked as 1 (i.e., evidence) in the second column of the table. The objective is to use DeepDive to find out the marginal probabilities (i.e., factual scores) that the other three counties would have high infection rate as well or not (marked as question marks in the table). Hence, we defined the following inference rule R with two predicates P1 and P2 in DeepDive:

*Also affiliated with University of Minnesota, MN, USA.

[1]This work is partially supported by the National Science Foundation, USA, under Grants IIS-1525953 and CNS-1512877.

```
P1: County X has high Ebola infection rate.
P2: Counties X and Y have same sanitation level.
R: If P1 & P2, then Y has high infection rate.
```

Given that the Montserrado county has a high Ebola infection rate, and it is on the same sanitation level as Margibi, Bong, and Gbarpolu counties, the inference in DeepDive used the input evidence data to report that Margibi, Bong, and Gbarpolu have high infection rates with factual scores 0.54, 0.52, and 0.63, respectively (second column in Figure 1(b)). Contrasting these factual scores with the ground truth of infection rate ranges of these four counties that are provided by the World Health Organization [44] (first column in Figure 1(b)), we consider the factual score of any county is correctly inferred if it is within the corresponding ground truth infection rate range. Then, by calculating the F1-score (i.e., the harmonic mean of precision and recall) of correctly inferred counties, DeepDive reported a low score of 0.39. This is mainly due to the fact that the rule did not acknowledge the spatial proximity of counties and its effect on the high infection rates. To remedy this issue within DeepDive, we added one more predicate (P3) and redefined the rule R to be:

```
P3: Counties X and Y are within 150 mi distance.
R: If P1 & P2 & P3, then Y has high infection rate.
```

With the new predicate, and feeding DeepDive with the locations of all the four counties per the map in Figure 1(c), DeepDive was able to adapt the factual scores of high Ebola infection rates in Margibi and Bong to be 0.51 and 0.45, respectively, as they are both within 150 miles from Montserrado, while reducing the factual score of Gbarpolu to be 0.06 as it is not within 150 miles from Montserrado. This example shows that the location information could significantly change the factual score in DeepDive. However, it also shows the obvious limitation of DeepDive when dealing with spatial predicates (e.g., P3). In particular, DeepDive treats any predicate as a boolean function, which yields either true or false (i.e., satisfied or not). So, although one can define spatial predicates in DeepDive (e.g., P3), internally DeepDive and its inference engine do not do anything special for spatial predicates. Due to this limitation, DeepDive has missed on the following two major issues: (1) Margibi county is significantly closer to Montserrado than Bong (Figure 1(c)), so, the factual score of Margibi should be significantly higher than Bong. However, DeepDive gives almost similar scores to both counties as they both satisfy P3. (2) Gbarpolu is only 160 miles from Montserrado, so, it should still have a good probability to be similar to Montserrado. However, DeepDive gives it a score that is close to 0 as it does not satisfy P3.

One interesting approach to simulate the spatial awareness in DeepDive is to generate rules that define the distance as a step function. For example, instead of having one rule R corresponding to the predicate `P3`, we can define a rule for each distance range (e.g., $10 <$ distance < 20, $20 \leq$ distance < 30, etc). However, as shown in Section VI, this comes with tremendous latency in the grounding step which makes it impractical to build knowledge bases.

Approach. In this paper, we present *Sya*; the first spatial MLN-based knowledge base construction system. *Sya* embeds the awareness of spatial relationships inside the *grounding* and *inference* phases of the knowledge base construction. In particular, *Sya* automatically generates a probabilistic model [43] that captures both *logical* and *spatial* correlations among its variables. Then, this model is used along with an efficient spatially-equipped statistical inference technique to infer the factual scores of knowledge base relations. In the above example, one can use *Sya* to redefine predicate P3 to be:

```
P3: The closer County Y to X, the higher its Ebola
infection rate.
```

With running this predicate, *Sya* was able to report the factual scores of Margibi, Bong, and Gbarpolu counties to be 0.76, 0.53, and 0.22, respectively. Given our ground truth knowledge, this result reports F1-score of 0.85, which is more accurate than what we get from DeepDive.

Challenges. *Sya* faces two main challenges in the grounding and inference phases, respectively. The *grounding challenge* is due to considering spatial correlations between all pairs of random variables associated with knowledge base relations. In case these variables are categorical with a large number of domain values h, the generated spatial correlations among each pair of variables will be of quadratic size in the number of domain values (i.e., $O(h^2)$). This can cause combinatorial explosion problems during the grounding operation [43], and later, the inference can become intractable. Thus, a pruning strategy is needed to ground only spatial correlations that will be effective in the inference phase. The *inference challenge* is the slow convergence to accurate factual scores in the presence of having spatial correlations among variables. In general, existing MLN-based systems require approximate inference techniques such as Gibbs sampling [46] to efficiently handle large probabilistic models. However, standard Gibbs sampling techniques depend on sequential updates of variables during sampling, which results in a significant latency overhead before convergence in case of having spatially-correlated variables as shown in [24]. Thus, a new efficient variation of Gibbs sampling is needed to handle these spatial correlations.

Contributions. Our technical contributions in this paper can be summarized as follows:

- We define *Sya* architecture, which can be used to extend any existing MLN-based knowledge base construction system and make it support spatial awareness (Section II).
- We extend a popular datalog-like language, namely DDlog [36], with spatial constructs that allow users to easily express their spatial semantics (Section III).
- We introduce a new spatial variation of the factor graph [43], namely *Spatial Factor Graph*, that is equipped with support for spatial correlations among variables. We also provide an optimization to heuristically prune inactive spatial correlations during grounding. This allows us to have a quality-scalability trade-off in *Sya* (Section IV).
- We introduce a new variant of Gibbs Sampling, namely *Spatial Gibbs Sampling*, that exploits the Conclique [23]

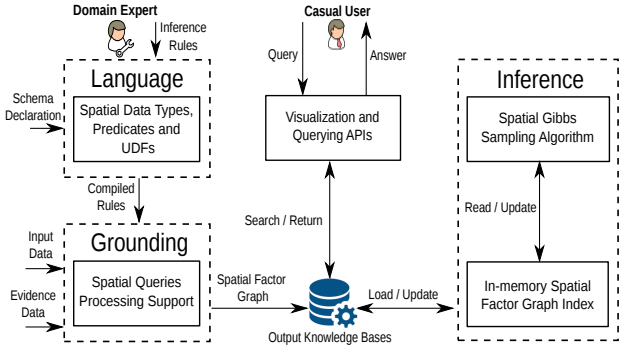


Fig. 2. *Sya* System Architecture.

concept from spatial statistics to efficiently sample from spatially-correlated variables. The proposed algorithm is extremely fast and has theoretical guarantees of convergence as shown in [24] (Section V).

- We perform an extensive evaluation of *Sya* with DeepDive [36] through building two real knowledge bases about the water quality in Texas [39], namely GWDB, and the air pollution in New York city [32], namely NYCCAS. The results show that *Sya* can achieve 120% and 70% higher F1-scores over DeepDive when building GWDB and NYCCAS, respectively, with at least 20% reduction in the execution time (Section VI).

II. SYA Architecture

Figure 2 gives the high-level system architecture of *Sya*. A domain expert would feed *Sya* with a set of inference rules, along with input and evidence data. A casual user can either use standard querying or visualization APIs to access the produced knowledge base relations with their factual scores. Internally, *Sya* is composed of three main modules, *language*, *grounding*, and *inference*, described briefly below:

Language module. This module extends a high-level declarative language, namely DDlog [36], with spatial data types (e.g., Point and Polygon), spatial predicates (e.g., Distance and Overlaps) and spatial UDFs (e.g., spatial objects extraction). This module allows domain experts to express the spatial semantics in the syntax of defining (1) the *schema* of database relations used, and (2) rules for *extracting* relations, and *correlating* them (i.e., inference rules). Once submitting the DDlog program, this modules checks the syntax correctness and the validity of used spatial constructs, compiles the program, and forwards it to the *grounding module*. Details of the language extensions are described in Section III.

Grounding module. This module receives the set of compiled rules from the *Language* module. Then, it evaluates the rules as spatial SQL queries (e.g., spatial join) against *input* (e.g., text and database relations) and *evidence* data. The output is a spatial variation of the factor graph [43] representing the knowledge base, and is stored in a relational database with spatial data support (e.g., PostGIS and MySQL Spatial). Details of this module are described in Section IV.

```
#Schema Declaration
S1: County (id bigint, location point, hasLowSanitation bool).
@spatial(exp)
S2: HasEbola? (id bigint, location point).

#Derivation Rule
D1: HasEbola(C1, L1) = NULL :- County(C1, L1, -).

#Inference Rule
R1: @weight (0.35)
HasEbola(C1, L1) => HasEbola(C2, L2) :- County(C1, L1, -), County(C2, L2, S2)
        [distance(L1, L2) < 150, within(liberia_geom, L1), S2 = true].
```

Fig. 3. Example on building EbolaKB using *Sya* Language.

Inference module. This module is triggered when it is required to estimate the factual scores (i.e., marginal probabilities) of knowledge base relations (i.e., variables in a factor graph). The module builds an in-memory pyramid index [3], referred to as *In-memory Spatial Factor Graph Index*, that partitions the spatial factor graph variables and correlations into a set of concliques [23], i.e., groups of non-neighboring spatial variables. Then, a novel Gibbs Sampling algorithm, referred to as *Spatial Gibbs Sampling*, is applied on the variables and correlations within each conclique to infer the factual scores of their corresponding relations. In case there is an update in the spatial factor graph, the in-memory index is updated through bulk insertion and deletion, and then the sampler is invoked on the concliques of the updated variables only. Details of this module are described in Section V.

III. The Language Module

Users of MLN-based knowledge base construction systems can use either native first-order clauses [16] (e.g., as in ProbKB [9], and Archimedes [10]) or high-level datalog-like languages (e.g., as in DeepDive [36] and SpannerLog [29]) to define the rules of constructing knowledge bases in a declarative manner. However, datalog-like languages have an advantage over native first-order rules in the integration with RDBMS engines and the ease of translating the rules syntax into equivalent SQL queries (details are in Section IV). In *Sya*, instead of providing a completely new language, we choose to extend the DDlog [36] language, a popular datalog-like language for encoding MLN probability distributions, with spatial data types, parameters, predicates, and user-defined functions (UDFs) to help users express the spatial semantics when building knowledge bases. Such extensions conform to the Open Geospatial Consortium (OGC) standard [33].

Relations and Rules in DDlog. DDlog allows its users to declare *typical* database relations to input/output data during the grounding and inference operations. It also supports a special type of *variable* relations, ended with a question mark in its declaration, to specify random variables. For example, the following statement declares a variable relation $Y?(s)$ based on a typical input relation $Data(s)$.

$$Y?(s) :- Data(s)$$

The statement defines a different binary random variable (taking either `True` or `False`) in $Y?(s)$ for each assignment

to s in $Data(s)$. Given variable relations, DDlog provides the ability to define *inference rules* that express the correlation among random variables in these relations. For example, the following weighted inference rule defines one logical bitwise-AND correlation for each entry in the output of equi-join between the variable relations X and Y on attribute s.

$$@weight(0.7) \quad X(r,s) \wedge Y(s) :- Z(r,s)[r = "a"]$$

The predicate $X(r,s) \wedge Y(s)$ is the *head* of the rule, and $Z(r,s)$ is the *body atom*. The body of the rule might contain a condition predicate, e.g. $[r = "a"]$ which filters the entries of relations based on the values that attribute r can take. The `@weight` parameter determines the confidence in the inference rule. Higher weights indicate higher confidence.

We describe the provided extensions by *Sya* in DDlog relations and rules using the example program in Figure 3, which builds the EbolaKB knowledge base in Section I.

Spatial Data Types. *Sya* adds four spatial data types, namely, `point`, `rectangle`, `polygon`, and `linestring`, to the schema declaration of relations in DDlog. For example, in Figure 3, each of the statements $S1$ and $S2$, which declare the input relation *County* and the variable relation *HasEbola*, respectively, has one spatial attribute of type `point`.

Spatial Variables and Correlation Specification. *Sya* allows its users to indicate which *variables* that we should consider their spatial attributes when inferring the factual scores of the knowledge base relations. A user can define the `@spatial(w)` parameter on the schema declaration of a variable relation to state that all instantiated variables in such relation should consider spatial correlations among themselves. Note that it is not allowed to annotate a variable relation with `@spatial(w)`, unless it has a spatial attribute (e.g., *HasEbola* in Statement $S2$ in Figure 3). The `w` input in `@spatial(w)` specifies the type of spatial weighing function used during the grounding and inference steps (details are in Section IV and V). This function could be either user-defined in the DDlog program or built-in in *Sya*. For example, the type `exp` in `@spatial(exp)` specifies an exponential distance weighing [2] function that is already implemented in *Sya*.

Spatial Predicates. *Sya* extends the body of DDlog rules with spatial predicates (e.g., `overlaps`, `within`, and `distance`) and functions (e.g., `union` and `buffer`) to support the evaluation of spatial queries in the grounding module (details are in Section IV). Spatial predicates can be composed. For example, the inference rule $R1$ in Figure 3, which indicates how neighboring Ebola infected counties affect each other, is composed of two spatial predicates `distance` and `within` that measure the distance between infected counties (using latitude and longitudes coordinates), and check whether they are located in Liberia or not, respectively.

Spatial User-defined Functions (UDFs). DDlog is powered with the ability to provide UDFs to specify feature extraction tasks that rely on integration with external tools (e.g., NLP preprocessing libraries). For spatial information, automatically extracting spatial entities (e.g., places) and relations from unstructured data can be challenging for end users. Therefore, *Sya* provides ready-to-use UDFs for spatial named entity recognition (NER), and objects extraction from unstructured text based on the GeoTxt library[2].

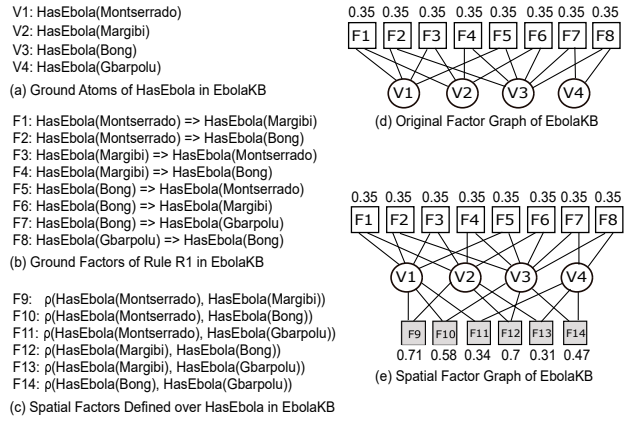


Fig. 4. Example on *Sya* Grounding for EbolaKB.

IV. The Grounding Module

The knowledge base construction rules represented by either native first-order clauses or datalog-like languages (as shown in Section III) can be viewed as a template for constructing the *probabilistic knowledge base model*, which encodes how knowledge base relations are linked to each other, and how their factual scores are correlated. This model is typically represented by a data structure, called factor graph [43]. A factor graph is a bipartite graph $\phi = \{\mathcal{V}, \mathcal{F}\}$ that has two sets of nodes: (1) a set of *random variables* $\mathcal{V} = \{v_1, v_2, ..., v_m\}$, and (2) a set of *factors* (a.k.a correlations) $\mathcal{F} = \{f_1, f_2, ..., f_n\}$, where each factor f_i is a function $f_i(\mathcal{V}_i)$ over a random vector $\mathcal{V}_i \subset \mathcal{V}$ indicating the correlation among the random variables in $\mathcal{V}_i$. Factors $\mathcal{F}$ together specify a joint probability distribution over all the random variables $\mathcal{V}$ in these factors.

Ground Factor Graph. The process of constructing the probabilistic knowledge base model as a factor graph is called *grounding*, and the output factor graph is referred to as a *ground factor graph*. In this process, we generate a random variable $v \in \mathcal{V}$ for each possible knowledge base relation and store it in a variable relation (e.g., *HasEbola* in Figure 3). The generated random variables are called *ground atoms*. Figure 4(a) shows an example of ground atoms in the EbolaKB example. We also generate a weighted factor $f \in \mathcal{F}$ for each possible grounding of an inference rule (e.g., rule $R1$ in Figure 3) that satisfies the predicates and conditions in the body of this rule. The generated factors are called *ground factors*. Figure 4(b) shows an example of ground factors of rule $R1$ in the EbolaKB example that satisfy the `distance` and `within` predicates. Figure 4(d) depicts an example ground factor graph based on ground atoms and factors from Figures 4(a) and 4(b), respectively. Each factor is represented by a square, and has edges with its variables

[2]https://github.com/geovista/GeoTxt

represented by circles. All factors are associated with the same confidence (i.e., weight) coming from the inference rule.

The joint probability distribution of a ground factor graph can be defined as follows:

$$P(\mathcal{V}=v) = \frac{1}{Z}\prod_{f_i \in \mathcal{F}} f_i(\mathcal{V}_i) = \frac{1}{Z}\exp\Big(\sum_{f_i \in \mathcal{F}} w_{f_i} n_{f_i}(v)\Big) \quad (1)$$

where n_{f_i} is the number of true groundings of factor f_i in variables assignment v, w_{f_i} is the weight of f_i, and Z is the partition function, i.e., normalization constant. Note that the distribution in Equation 1 represents the marginal inference, which is commonly used in the knowledge base literature.

In this section, we describe how *Sya* extends the ground factor graph to support spatially-correlated ground atoms (Section IV-A). In addition, we discuss the database support in *Sya* for constructing the factor graph in an efficient manner (Section IV-B). Finally, we provide an optimization to prevent the combinatorial explosion that could happen during the grounding of spatial factor graph (Section IV-C).

A. Spatial Factor Graph

In MLN-based applications, the correlations between variables, which are knowledge base relations in our case, are captured in the factor graph using logical factors such as bitwise-OR and imply. However, in the case of having variables representing spatial phenomena (e.g., epidemiology), logical correlations are not enough to obtain accurate inference scores for these variables. In fact, ground atoms from the same type of spatial variable tend to have high spatial correlation among each other (e.g., *HasEbola(Margibi)* and *HasEbola(Bong)*). This is one of the fundamental properties of spatial analysis, where "everything is related to everything else, but nearby things are more related than distant things". We refer to these ground atoms as *spatial ground atoms*.

A main limitation in using existing inference rules to capture the spatial correlations between spatial ground atoms is that there is no efficient way to represent the weight of the rule as a function of distance between atoms. Existing MLN-based knowledge base systems provide only two options to specify weights in inference rules. The first option is to fix weights as constants (e.g., the inference rule $R1$ in Figure 3). However, in this option, we need to have a separate inference rule for each possible distinct value of distance, which is impractical. For instance, in the EbolaKB example, we would need to define a new inference rule $R2$ similar to $R1$, but, with weight of 0.5 if the distance between two counties is less than 100, and so on. The second option is to learn distinct weights for different distance values based on training data. However, this option requires enough training data available for all possible distance values, which is impractical as well.

In *Sya*, we introduce a new type of factors, called *spatial factors*, to capture the spatial correlations among spatial ground atoms. Such factors are generated for each possible pair of ground atoms from the same type of spatial variable and assigned proper weights based on the relative distance among atoms. We first provide a definition for spatial factors over ground atoms coming from *binary* spatial variables, then we extend this definition for the case of *categorical* variables.

Definition 1: Given two spatial ground atoms v_j and v_k of a binary spatial variable, and a spatial weight $w_{d(v_j,v_k)}$ based on the distance $d(v_j, v_k)$ between v_j and v_k, a spatial factor $\rho_{j,k}$ over v_j and v_k is a multi-valued function, where

$$\rho_{j,k} = \begin{cases} e^{w_{d(v_j,v_k)}} & v_j = v_k \\ e^{-w_{d(v_j,v_k)}} & otherwise \end{cases} \quad (2)$$

As shown in Equation 2, spatial factors favor similar values of close ground atoms (i.e., spatial clustering), where each factor specifies a *unique* weight based on the distance between involved atoms. Generally, spatial correlations can be defined on more than two grounds. However, we focus only on binary correlations. The extension to high-order cases is intuitive as well, but, out of scope of this paper.

We propose the spatial factor $\rho_{j,k}$ in an exponential form to easily extend the probability distribution $P(\mathcal{V}=v)$ in Equation 1 by directly adding the spatial weight $w_{d(v_j,v_k)}$ as a new potential function to the existing ones (i.e., $\sum_{f_i \in \mathcal{F}} w_{f_i} n_{f_i}(v)$). Formally, given a set of spatial factors ρ, we extend the factor graph $\phi = \{\mathcal{V}, \mathcal{F}\}$ to be a *spatial factor graph* $\mathcal{G} = \{\mathcal{V}, \beta\}$, which has the same set of random variables $\mathcal{V}$, and a combined set of non-spatial and spatial factors $\beta = \mathcal{F} \cup \rho$. As a result, the equivalent probability distribution $P(\mathcal{V}=v)$ to the spatial factor graph $\mathcal{G}$ becomes:

$$P(\mathcal{V}=v) = \frac{1}{Z}\exp\Big(\sum_{f_i \in \mathcal{F}} w_{f_i} n_{f_i}(v) + \sum_{\rho_{j,k} \in \rho} w_{d(v_j,v_k)}(\mathbb{1}_{v_j=v_k} - \mathbb{1}_{v_j \neq v_k})\Big) \quad (3)$$

where $\mathbb{1}_{v_j=v_k}$ and $\mathbb{1}_{v_j \neq v_k}$ are indicator functions. Figure 4(e) depicts an example spatial factor graph for EbolaKB after adding the spatial factors defined over $HasEbola$ atoms.

In *Sya*, there is no need to define inference rules for spatial factors. These factors are automatically generated for variables that are annotated with the `@spatial(w)` keyword in their schema declaration, where the input `w` determines how to calculate the weight $w_{d(v_j,v_k)}$. For example, the type `exp` in `@spatial(exp)` defined over statement $S2$ in Figure 3 indicates that $w_{d(v_j,v_k)}$ should be calculated using exponential distance weighing [2] function. Figure 4(c) shows an example of grounding the spatial factors (highlighted with gray) that are defined over $HasEbola$ variables.

Spatial Factors for Categorical Variables. In case of having knowledge base relations represented with a categorical variable (i.e., a variable with h possible domain values), the grounding process generates h instances of the ground atom corresponding to each knowledge base relation, where each instance indicates whether one possible domain value is selected or not [36]. As a result, we adapt the spatial factor

function in Equation 2 to be defined over a pair of instances from two spatial ground atoms as follows:

***Definition** 2:* Given two spatial ground atoms v_j and v_k of a categorical spatial variable with h domain values, and a spatial weight $w_{d(v_j,v_k)}$ based on the distance $d(v_j, v_k)$ between v_j and v_k, a spatial factor $\rho_{j,k}(t_j, t_k)$ over the instance of v_j for domain value t_j, namely $v_j(t_j)$, and the instance of v_k for domain value t_k, namely $v_k(t_k)$, is a multi-valued function, where

$$\rho_{j,k}(t_j, t_k) = \begin{cases} e^{w_{d(v_j,v_k)}} & v_j(t_j) = v_k(t_k) = 1, t_j = t_k \\ e^{-w_{d(v_j,v_k)}} & v_j(t_j) = v_k(t_k) = 1, t_j \neq t_k \\ 1 & otherwise \end{cases} \tag{4}$$

Similar to Equation 2, Equation 4 favors similar domain values of close ground atoms. In case the value of either $v_j(t_j)$ or $v_k(t_k)$ is 0, we refer to $\rho_{j,k}(t_j, t_k)$ as an *inactive* spatial factor, because the factor value will be 1 and will not have any effect on the joint probability distribution. Note that the joint probability distribution can be extended in the categorical case similar to Equation 3. Since we have h instances for each of the two ground atoms v_j and v_k, we end up with h^2 spatial factors between v_j and v_k. This results in a combinatorial explosion problem during the execution of grounding. More details on this issue are in Section IV-C.

B. Rules Translation and Execution

Existing MLN-based knowledge base construction systems (e.g., [9], [36]) efficiently construct the factor graph by evaluating its corresponding inference rules as SQL queries to exploit the DBMS scalability and efficiency. As a result, *Sya* provides a spatial rules-queries translator and a database driver to evaluate the spatial extensions to these rules (shown in Section III) as spatial SQL queries as well.

Spatial Rules-Queries Translator. Typically, the inference rules are translated into a set of inner and outer join queries with simple predicates to check (e.g., equality and range checks). *Sya* extends this translation process with support for two spatial queries; *spatial join* and *range query*. In case of having a rule with a spatial predicate, e.g., `distance`, *Sya* reroutes its translation into these spatial queries rather than the original join queries. Moreover, *Sya* provides two effective optimizations: (1) It supports creating on-fly spatial indices (e.g., R-tree [20] and GIST [21]) on relations with spatial attributes, making the evaluation of complex predicates (e.g., `overlap`) is efficient. (2) It provides a simple heuristic query optimizer that re-orders the execution of nested spatial queries that come from rules with multiple spatial predicates. Figure 5 shows an example of translating the inference rule $R1$ from Figure 3, which has two spatial predicates `distance` and `within` that are translated into a spatial join and range query, respectively. Note that, although the `distance` predicate comes before the `within` one in the rule, *Sya* re-orders their translated queries to have the range query runs before the spatial join to reduce the number of tuples to be joined.

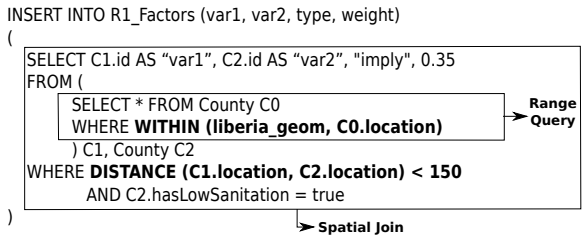


Fig. 5. Example on Rules Translation in *Sya*.

Integration with Spatial Databases. *Sya* fully integrates with scalable spatial database engines, e.g., PostGIS, and MySQL Spatial to execute the translated queries. Such engines support both spatial and non-spatial queries. Thus, SQL queries corresponding to rules with non-spatial predicates can still be executed on them. In addition, *Sya* provides an abstract database driver that supports defining the spatial storage, functions and query capabilities needed to ground spatial factor graphs. Such abstract can be extended by users to run their spatial database engine choice inside *Sya*.

C. Scaling Up the Grounding of Spatial Factor Graph

The number of spatial factors ρ can easily explode when dealing with categorical variables that have large domains (i.e., the number of domain values h is large) (details are in Section IV-A). This can significantly affect the scalability of the knowledge base construction process. As a result, we introduce an optimization for pruning the spatial factors that are more likely to be *inactive* based on co-occurrence statistics of their corresponding domain values in the input evidence data. Basically, for each pair of domain values (i, j) of a spatial categorical variable v, if these values co-occur with certain probabilities that exceed a pre-defined threshold T in the evidence input data, then we generate a spatial factor $k(i, j)$ over this pair of values. In case not passing the threshold T, we ignore all spatial factors defined over this pair of values as they are considered inactive. Using Bayesian analysis, we estimate the co-occurrence probabilities of (i, j) in two parts: $P(i|j)$ and $P(j|i)$, where

$$P(i|j) = \frac{\text{no. of } i \text{ and } j \text{ appear together in evidence data}}{\text{no. of } j \text{ appers in evidence data}}$$

and, similarly,

$$P(j|i) = \frac{\text{no. of } i \text{ and } j \text{ appear together in evidence data}}{\text{no. of } i \text{ appers in evidence data}}$$

Note that the threshold T should be tuned by *Sya* users. We discuss the effect of T on the performance of *Sya*, and show its scalability-quality trade-off in Section VI.

V. The Inference Module

The main objective of the inference step is to estimate the marginal probabilities of variables (i.e., ground atoms) in the factor graph. In our case, such probabilities are considered the output factual scores of the knowledge base relations. To

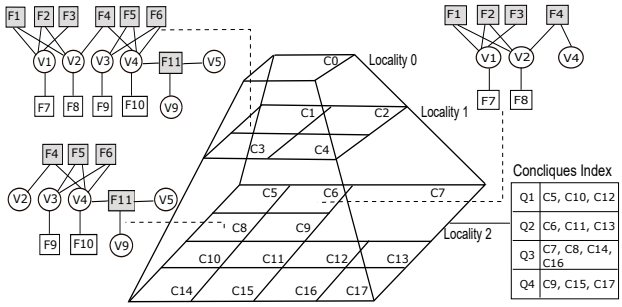


Fig. 6. Example on In-memory Pyramid Index of Spatial Factor Graph.

perform this step in MLN-based knowledge base construction systems, approximate inference via Gibbs sampling is commonly used [10], [28], [46], [47]. However, using existing variations of Gibbs sampling to infer from the spatial factor graph (i.e., factor graph with spatial factors) is inefficient, because the sampling nature in these algorithms relies on single-site, or sequential, updates within the same inference epoch. This, in turn, raises the need for a large number of iterations (i.e., slow convergence) to obtain an acceptable output when there are some variables that are spatially-correlated as shown in [23]. In this section, we provide a new Gibbs sampling algorithm, namely *Spatial Gibbs Sampling*, that overcomes this limitation by employing efficient spatial statistics and in-memory access techniques to guarantee the rapid convergence in case of having spatially-correlated variables.

Main Idea. State-of-the-art parallelized Gibbs sampling algorithms [46], [47] randomly partition the variables into a set of buckets and then sample these buckets in parallel. Even though these algorithms will finish the sampling iterations faster than the sequential ones, they may not converge to an acceptable solution as spatially-dependent variables might run in parallel (i.e., independent of each other). This will force the sampler to run additional inference epochs to converge, and hence incur a significant latency overhead. Another solution is to use block-based Gibbs sampling (e.g., [42]). However, this solution requires joint sampling at each block, which is computationally-inefficient as well.

In *Sya*, we devised an approach that combines in-memory spatial partitioning technique, namely pyramid index [3], with a well-known spatial statistics concept, namely concliques [23], to heuristically partition the spatial factor graph into a set of spatially-independent partitions. We refer to this way of partitioning as *concliques-based partitioning*. The resulting partitions can be sampled in parallel to each other using standard Gibbs sampling. It is theoretically proven that concliques-based partitioning makes Gibbs sampler converge faster than traditional random partitioning [24]. First, we give the details of the pyramid index and concliques concepts. Then, we provide an algorithm that exploits such concepts to provide our proposed *spatial Gibbs sampling* algorithm.

In-memory Spatial Factor Graph Index. *Sya* employs an in-memory partial pyramid index [3] to spatially partition the spatial factor graph. The pyramid index decomposes the whole space into L locality levels (i.e., pyramid levels), where the space in level l is partitioned into 4^l grid cells. In each cell, *Sya* stores a pointer-based index to the spatial ground atoms - along with their connected factors - that have locations contained in the cell's spatial region. A spatial ground atom v may contribute to up to $L-1$ pointer-based indices: one per each locality level starting from level 1 to the lowest maintained grid cell containing the v's location. The root level (Cell 0) of the pyramid has no spatial relationships between atoms. In addition, a factor node can be duplicated if it is connected to more than one atom at different cells.

Since the pyramid index is a hierarchical space partitioning technique, it guarantees to completely cover any given space and allows *Sya* users to control the size of neighbourhood. A locality level `l` acts like a "zoom" level (e.g., city block, entire city). Another advantage of the pyramid index is its ability to store data in non-leaf cells (i.e., cells that are not at the lowest pyramid level), which helps in storing the spatial factor graph efficiently at the different pyramid levels. Figure 6 shows an example pyramid index of a spatial factor graph. The index is assumed to have 3 levels only, where there are empty cells due to not having variables contained in these cells. We show the partitioning details of partial factor graph in cells C_1, C_6 and C_8. Note that the partial graph at C_1 is divided into two sub graphs at C_6 and C_8 because C_6 and C_8 are children of C_1. Also, factor node F_4 is replicated in both C_6 and C_8 because it is connected to V_2 and V_4 which are at different cells.

Initially, to build the pyramid, all spatial ground atoms are used to build a complete pyramid of height L, such that all cells in all L levels are present and contain a partial graph. The initial height L is chosen according to the level of locality desired. Once the initial build is done, a merging step is called to scan all cells starting from the lowest level and merge quadrants (i.e., four cells with a common parent) into their parent if three of these quadrants are empty. Once an incremental update is received, *Sya* performs a sequence of splitting and merging operations over the pyramid cells, if necessary. A cell is split only if it is over a capacity threshold and splitting its contents spans at least two children cells.

Concliques-based Partitioning. A conclique is defined as a set of locations such that no two locations in this set are neighbours [23]. For example, the cells of locality level 2 in Figure 6 can be divided into four concliques: $\mathcal{Q}_1 = \{C_5, C_{10}, C_{12}\}$, $\mathcal{Q}_2 = \{C_6, C_{11}, C_{13}\}$, $\mathcal{Q}_3 = \{C_7, C_8, C_{14}, C_{16}\}$ and $\mathcal{Q}_4 = \{C_9, C_{15}, C_{17}\}$. The main idea behind defining concliques is ensuring the neighbouring independence between variables in the same conclique set, and hence these variables can be sampled in parallel. Assume there is a spatial factor graph defined over the whole cells in the locality level 2 of Figure 6. The sampling process over these cells can be done using four iterations. The first iteration handles conclique Q_1 by initiating three threads to process C_5, C_{10} and C_{12} in parallel. In each thread, we sample the variables of its associated cell sequentially using standard Gibbs sampling. After sampling cells in Q_1 is done, the second, third and fourth iterations can be done sequentially to handle Q_2, Q_3 and Q_4, respectively.

Algorithm 1 Function SPATIALGIBBSSAMPLING (SpatialFactorGraph $\mathcal{G}$, Instances K, Epochs E)

```
1: C ← Null /* Sampling Counters */
2: for all v ∈ V do in parallel
3:     C[v] ← 0
4: e ← E/K /* No. of Epochs Per Instance*/
5: P ← BUILDPYRAMIDINDEXOFSPATIALFACTORGRAPH (G)
6: Q ← BUILDCONCLIQUESOFPYRAMIDINDEX (P)
7: L ← No. of Levels in P
8: while e ≠ 0 do
9:     for all k ∈ {1, 2, ..., K} do in parallel
10:        for all l ∈ {2, 3, ..., L − 1} do serially
11:            T ← GETNONEMPTYCELLS (P, l)
12:            U ← GETMINCONCLIQUESCOVER (Q, l, T)
13:            for all u ∈ U do serially
14:                for all t ∈ T ∩ u do in parallel
15:                    C_k[V_t] ← RUNSTANDARDGIBBSSAMPLER (V_t, G, C_k)
16:    C ← (Σ_{k=1}^{K} C_k)/K, e − −
17: end while
18: for all v ∈ V do in parallel
19:    v.Prob ← CALCMARGINALPROBABILITY (C, v)
```

Algorithm. Algorithm 1 depicts the pseudo code for the spatial Gibbs sampler that takes the following three inputs: the spatial factor graph G, the number of running instances K that can run in parallel, and the number of inference iterations E. The algorithm keeps track of the current counts of sampled values in each variable $v \in \mathcal{V}$ through variable C, initialized by zeros. The algorithm then starts by computing the number of inference epochs that can be handled per each running instance and stores it in variable e. Note that e represents the actual number of inference epochs that run sequentially because different inference instances execute in parallel. Each of these inference instances then starts to process one inference epoch in parallel (i.e., K inference epochs are running simultaneously). Then, the algorithm builds (1) a pyramid index of the input spatial factor graph, referenced by variable P, and (2) an index of concliques for each level in the pyramid index, referenced by variable Q (Lines 5 and 6).

In each inference epoch (Lines 10 to 15), the algorithm first traverses each pyramid level l, and gets the minimum set of concliques U that cover the partial spatial factor graphs in this level l (Lines 11 to 12). For example, the locality level 2 in Figure 6 has two partial graphs at C_6 and C_8 cells. Then, the algorithm will return Q_2 and Q_3 as minimum set of covering concliques. After that, for each conclique $u \in U$, the algorithm processes the non-empty cells (i.e., that have partial graphs), associated with u in parallel. In the running example, the algorithm starts with conclique Q_2, which has only cell C_6 to process. After finishing Q_2, the algorithm processes Q_3 which has only cell C_8. At each cell t, the algorithm sequentially samples all variables in t using a standard Gibbs sampler. In our experiments, we used the variation of Gibbs sampling inside DeepDive [36] as it is computationally-efficient, easy-to-implement, and can support incremental inference. Note that by traversing different pyramid levels, the algorithm might sample the same variable multiple times (i.e., it happens that one variable is connected with two factors at different locality levels). However, this situation will not harm the validity of results as shown in block-based Gibbs sampling algorithms [42]. In addition, it will not significantly increase the latency overhead compared to the huge performance gain achieved from processing the cells in each conclique in parallel.

After all inference instances finish their current inference epoch, we set the values of C with the average of obtained counts of samples from these instances (Line 16) and then proceed to another inference epoch with the new counts. We repeat this process e times, and then use the final counts of samples to calculate and update the marginal probability of each variable as in [43](Lines 18 and 19).

Complexity. The complexity of Algorithm 1 can be estimated as $O(L|\mathcal{V}| + L + (\frac{E}{K})(\frac{4}{3})(1-(\frac{1}{4})^{L+1})|\mathcal{V}|^2)$ where $O(L|\mathcal{V}|)$ is the cost of building the pyramid index (Line 5), $O(L)$ is the cost of building the concliques in all pyramid levels (Line 6), and $O((\frac{E}{K})(\frac{4}{3})(1-(\frac{1}{4})^{L+1})|\mathcal{V}|^2)$ is the cost of applying the Spatial Gibbs Sampling steps (Lines 8 to 19). The complexity can be approximated to be $O(L|\mathcal{V}| + (\frac{E}{K})|\mathcal{V}|^2)$. Since the value of $\mathcal{V}$ is significantly larger than L, the complexity can be further approximated to be $O((\frac{E}{K})|\mathcal{V}|^2)$.

System	No. Rels	No. Rules	No. Vars	No. Factors
GWDB	1	11	104K	39.5M
NYCCAS	1	4	34K	233K

TABLE I
STATISTICS OF KBS USED IN EXPERIMENTS.

VI. EXPERIMENTS

In this section, we experimentally evaluate the quality and scalability of *Sya*, based on a real system implementation [35] inside DeepDive [36]. We choose DeepDive as it is one of the most popular probabilistic knowledge base construction systems, with many success stories in vital applications (e.g., fighting human trafficking). In addition, DeepDive provides an open-source implementation for both the grounding and inference phases[3]. We compare the performance of *Sya* with DeepDive while building two real knowledge bases. We also extensively investigate the quality and convergence of *Sya* under different system parameters.

A. Experimental Setup

Datasets. In our experiments, we have built two knowledge base systems, namely GWDB and NYCCAS, using both *Sya* and DeepDive. Table I illustrates the different statistics of these systems including the number of input database relations (No. of Rels), the number of inference rules (No. Rules) used to build the knowledge bases, the number of variables (No. Vars) and factors (No. Factors) in the generated factor graphs.

The *GWDB* system builds a knowledge base about the water quality in Texas. The input to this system is the Texas Ground Water Database (GWDB) relation [39], which is collected by Texas Water Development Board (TWDB) about 9831 water wells. It contains information about each well such as location, depth and the concentration of different elements such as

[3]https://github.com/HazyResearch/deepdive

```
Sya Syntax
Well (id bigint, location point, arsenic_ratio double).

@spatial(exp)
IsSafe? (id bigint, location point).

@weight(0.7)
R1: IsSafe(W1, L1) => IsSafe(W2, L2) :- Well(W1, L1, R1), Well(W2, L2, R2)
        [distance(L1, L2) < 50, R1 < 0.2, R2 < 0.2].

DeepDive Syntax
Well (id bigint, loc_x double, loc_y double, arsenic_ratio double).
Distance (id1 bigint, id2 bigint, dist double).

function calc_distance over (id1 bigint, loc_x1 double, loc_y1 double,
                             id2 bigint, loc_x2 double, loc_y2 double)
                         returns rows like Distance
                         implementation "udf/calc_distance.py" handles tsj lines.
Distance+= calc_distance (W1, L1_x, L1_y, W2, L2_x, L2_y):-
                        Well(W1, L1_x, L1_y, -), Well(W2, L2_x, L2_y, -).

IsSafe? (id bigint, loc_x double, loc_y double).

@weight(0.7)
R1: IsSafe(W1, L1_x, L1_y) => IsSafe(W2, L2_x, L2_y) :-
      Well(W1, L1_x, L1_y, R1), Well(W2, L2_x, L2_y, R2), Distance(W1, W2, D)
        [D < 50, R1 < 0.2, R2 < 0.2].
```

Fig. 7. Example on a Rule for the GWDB KB in *Sya* and DeepDive.

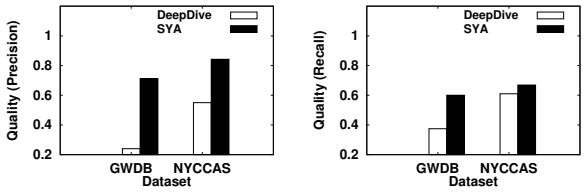
fluoride and arsenic. We developed a program that consists of 11 inference rules that infers the risk of drinking from each well. For example, a certain well is considered dangerous if the arsenic concentration exceeded a certain threshold defined by the Environment Protection Agency and its location is near from another risky well.

The *NYCCAS* system builds a knowledge base about the air pollution concentrations in the New York city. The input data is mainly a raster database relation maintained by the department of Health and Mental Hygiene (DOHMH) [32] about the annual predicated concentrations for specific elements in the air. Unlike the GWDB system, we developed a smaller program which has 4 inference rules only that relate different guidelines from the Environment Protection Agency about the air pollution with the observations from raster data. Note that the factor graph statistics for NYCCAS are relatively small compared to GWDB, and both have one input relation only.

In both systems, ground truth information (i.e., evidence data) is available for all extracted knowledge base relations. In addition, each variable has binary domain values. We will increase the number of domain values only when we study the effect of the pruning threshold T.

Rules. To have a fair comparison when building these knowledge bases, we submitted two equivalent DDlog programs to both *Sya* and DeepDive. Figure 7 shows an example on an inference rule $R1$ used to develop the GWDB knowledge base in both *Sya* and DeepDive. This rule indicates that the closer a well to another safe well that has low arsenic level, the higher probability this well becomes safe. As shown in the figure, we used our spatial extensions of DDlog to express the spatial semantics in *Sya* rules. In case of DeepDive, we provided an equivalent user-defined function implementation to the basic spatial functions. In the shown example, we defined the `calc_distance` function that calculates distances between all possible pairs of wells. All calculated distances are materialized to be used along with the inference rule.

Evaluation Metrics. In all experiments, to measure the scalability, we use the running times of the grounding and inference phases. To measure the quality of factual scores, we use the following three metrics: 1) *Precision* ($Prec$): the number of predicted factual scores that match the ground truth within 0.1 error (i.e., correctly inferred scores), over the total number of factual scores to be predicted. 2) *Recall* (Rec): the number of correctly inferred scores (calculated similar to $Prec$), over the total number of factual scores that should be predicated correctly according to the evidence data. 3) *F1-score*: the harmonic mean of precision and recall, which is calculated as $2(Prec * Rec)/(Prec + Recall)$.



(a) Dataset vs. Precision (b) Dataset vs. Recall

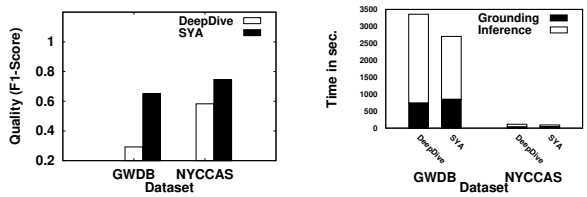
Fig. 8. Comparison with DeepDive (Precision and Recall)

Environment. Both systems are implemented in C++. We run all experiments on a single machine with Ubuntu Linux 14.04. Each machine has 8 quad-core 3.00 GHz processors, 64GB RAM, and 4TB hard disk. We use PostgreSQL, and its spatial extension PostGIS, to execute SQL queries.

Parameters. Unless otherwise mentioned, we set the number of inference epochs to 1000, the input of the `@spatial` parameter (Section III) to the exponential distance weighing function [2], and the pruning threshold T to 0.5. In *Sya*, we built a pyramid index for both Texas state and New York city. In each index, the number of pyramid levels L is 8, and the locality level l is the lowest pyramid level (i.e., 8).

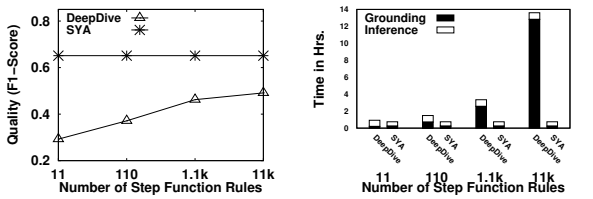
B. Experimental Results

1) Comparison with DeepDive using Different Datasets: Figure 8(a) shows the precision results obtained by *Sya* and DeepDive while building the GWDB and NYCCAS knowledge bases. Due to the probabilistic nature of the sampling algorithms, we run all inference rules for both systems 5 times, and after each run, we report the quality of the system measured by the precision. Then, we average the obtained scores for each system (we follow the same approach in all precision and recall experiments in the paper). As shown in the figure, *Sya* outperforms DeepDive significantly with relative precision improvements of more than 53% in both datasets. The main reason behind the impressive performance of *Sya* is that the factual scores, in each of the two knowledge bases, have spatial correlations among each other, which is a common property in all spatial applications. These correlations were properly utilized inside *Sya* using the spatial factors, and hence results in more accurate factual scores. We also notice that the variance between the precision values of *Sya* in both datasets is significantly smaller than DeepDive. This verifies our hypothesis that dealing with spatial predicates as a boolean



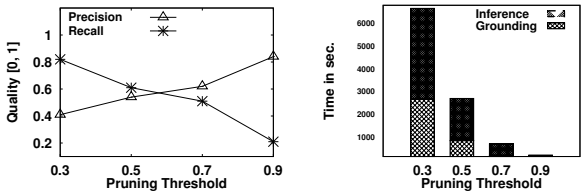
(a) Dataset vs. F1-Score (b) Dataset vs. Exec. Time

Fig. 9. Comparison with DeepDive (F1-Score and Execution Time)



(a) No. of Rules vs. F1-Score (b) No. of Rules vs. Exec. Time

Fig. 10. Comparison with DeepDive using Step Function Rules



(a) Threshold vs. Quality (b) Threshold vs. Exec. Time

Fig. 11. Effect of Pruning Threshold on Quality and Execution Time

function, as in DeepDive, leads to inaccurate results. Recall the EbolaKB example in the introduction, when Gbarpolu county was only 10 miles more than the cut-off threshold, and yet, it got a score that is close to 0.

Figure 8(b) shows the recall results obtained by *Sya* and DeepDive while building the GWDB and NYCCAS knowledge bases. For the GWDB dataset, we still have the same conclusion that *Sya* is better than DeepDive. In this case, the improvement ratio is around 60%. For the NYCCAS dataset, we notice that *Sya* still has higher recall output, yet, with a small improvement ratio of 9%. This is because the NYCCAS dataset has a significant amount of its evidence data entries that follow random assignments. This limits the recall of *Sya* and makes it close to DeepDive.

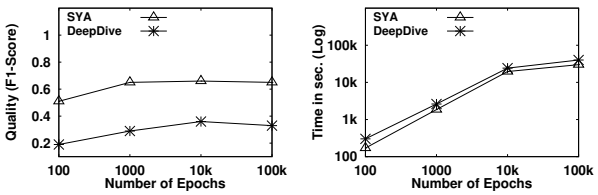
Figure 9(a) shows the F1-score for both *Sya* and DeepDive while building the GWDB and NYCCAS knowledge bases. For the two knowledge bases, *Sya* were able to significantly increase the F1-score compared to DeepDive. Specifically, *Sya* has an F1-score improvement of 120% and 27% over DeepDive in GWDB and NYCCAS, respectively. We can conclude from the results of the three quality metrics that the effect of considering the spatial correlations while inferring the factual scores is huge and can significantly boost the quality of the knowledge base outputs.

Figure 9(b) shows the grounding and inference times for both *Sya* and DeepDive while building the GWDB and NYCCAS knowledge bases. As seen in the figure, the grounding time of *Sya* is at maximum 15% higher than DeepDive in both datasets due to the additional overhead of generating spatial factors. We also observe that *Sya* has at least 30% reduction in the inference time in both datasets. The main reason behind this performance gain is applying the concliques-based partitioning in the spatial Gibbs sampling algorithm (Section V), which enables the parallel sampling for all variables within the same conclique. Note that the grounding and inference times of both systems are significantly low in NYCCAS compared to GWDB because of the small size of the factor graph, however, *Sya* still has the same improvement ratio.

2) Comparison with DeepDive using Step Function Rules: In this experiment, we compare the performance of *Sya* with DeepDive while using a step function in DeepDive to generate a set of inference rules that approximate the spatial effect. For example, we can use a step function to replace the inference rule $R1$ in Figure 7 by the following set of range-based rules: Rule $R1(1)$ that defines `@weight(0.9)` for distance range $0 \leq D < 10$, Rule $R1(2)$ that defines `@weight(0.8)` for distance range $10 \leq D < 20$, etc. Note that large weights are associated with small distance values. Figure 10(a) shows the F1-score for both *Sya* and DeepDive while varying the number of generated step function rules in DeepDive from 11 to $11k$. We report the results for the GWDB knowledge base only. By increasing the number of generated rules, we obtain more accurate weights to be associated with the inference rules, and hence achieve better F1-scores. However, as shown in Figure 10(b), this comes with high latency in the grounding phase as the number of generated SQL queries becomes large as well (i.e., one SQL query per rule). For example, generating $11k$ step function rules, instead of the original 11 rules of GWDB, requires more than 12 hours in the grounding phase to obtain 20% less F1-score compared to *Sya*, which is the best score achieved by DeepDive in our experiments.

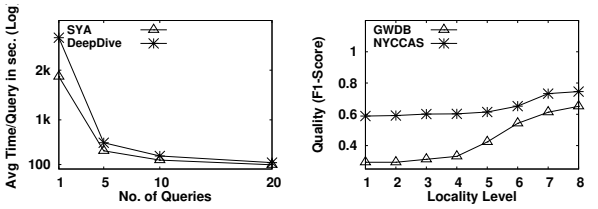
3) Effect of Pruning Threshold: Figure 11(a) shows the effect of changing the pruning threshold T on the precision and recall of *Sya*. In this experiment, we report the results of the GWDB knowledge base only. However, the same findings apply on the NYCCAS dataset. We changed the number of domain values of the generated relations to be 10 instead of 2. This means that the number of spatial factors between any pair of relations (i.e., ground atoms) is 100. By ranging the value of T from 0.3 to 0.9, we obtain a trade-off between the precision and recall results. When the value of T is small, the range of allowed domain values is widened, and hence the recall value becomes higher, and vice versa. For the precision case, by increasing the value of T, we keep only the spatial factors that are likely to be effective in capturing the spatial correlation, and hence the probability of having accurate results becomes higher. This results in higher precision values.

Figure 11(b) shows the effect of changing the pruning threshold T on the grounding and inference times of *Sya*. Obviously, increasing the value of T results in a less number of



(a) Epochs vs. F1-Score (b) Epochs vs. Inference Time

Fig. 12. Effect of Inference Epochs on F1-Score and Inference Time



(a) Incremental Inference (b) Locality Level vs. Quality

Fig. 13. Effect of Incremental Inference and Locality Level

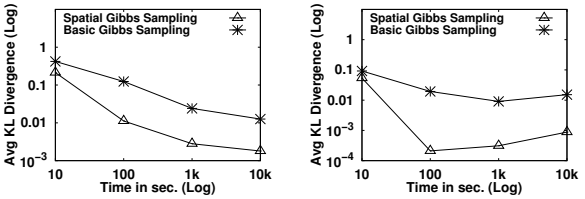
spatial factors to be processed in both grounding and inference phases, and hence the total running time drops significantly. For example, by changing the value of T from 0.3 to 0.9, the improvement ratio of total running time becomes 96%. However, this might come with the cost of less recall results as shown in Figure 11(a).

4) Effect of Number of Inference Epochs: Figure 12(a) shows the effect of changing the number of inference epochs on the quality of *Sya* and DeepDive. We report the results for the GWDB knowledge base. We change the number of epochs from 100 to 100k, while observing the F1-score for both systems. We find that increasing the number of epochs allows both systems to converge towards more accurate results, until a threshold. The quality of both systems started to saturate around 1000. Yet, we find that the difference in quality scores at 10k and 100k compared to 1000 is higher in DeepDive than *Sya*. For *Sya*, the average difference is 0.01. While it becomes 0.04 in case of DeepDive. Note that *Sya* is consistently better than DeepDive regardless the number of epochs.

Figure 12(b) shows the effect of changing the number of inference epochs on the inference time, reported in a log-scale, of both *Sya* and DeepDive. We use the same experiment setup in Figure 12(a). We can observe that *Sya* is still faster than DeepDive in both small and large number of epochs, yet, both systems are still within the same order of magnitude. The improvement ratio of *Sya* over DeepDive ranges from 20% to 31% at maximum. This confirms the inference running time results in Figure 9(b). We have also tried to re-run the same experiment with different order of variables in the factor graph. However, we got very similar numbers. This shows that Gibbs sampler, in both standard and spatial variants, is still very practical even though it has no guarantees of convergence.

5) Effect of Incremental Inference and Locality Level: Figure 13(a) shows the effect of supporting the incremental inference on the performance of both *Sya* and DeepDive while building the GWDB knowledge base. In this experiment, we start with applying the inference on the whole factor graph nodes. Then, we gradually change the values of some nodes (i.e., query nodes), and calculate the corresponding average time to finish the inference over these changed nodes. We vary the number of changed nodes from 1 to 20. As we can see, the incremental inference in *Sya* takes 40% less time than DeepDive to finish the whole queries. Since most of the changed nodes are spatially-correlated from the application nature, *Sya* has a better chance to rapidly converge more than DeepDive. This is because of the spatial support that *Sya* injects in the Gibbs sampling approach.

Figure 13(b) shows the quality of *Sya* in building GWDB and NYCCAS knowledge bases while varying the locality level (i.e., pyramid level) from 1 to 8. In general, both cases show that the F1-score of *Sya* increases when it uses more localized pyramid cells. However, the localization has more influence on GWDB than NYCCAS. This behaviour further verifies that just providing precise locality level, while fixing other parameters, could result in higher quality factual scores.



(a) GWDB Dataset (b) NYCCAS Dataset

Fig. 14. Quality of Spatial Gibbs Sampling with Different Datasets

6) Quality of Spatial Gibbs Sampling: In this experiment, we directly compare the quality of our proposed *spatial Gibbs sampling* with the state-of-the-art Gibbs sampling [46], [47], that has been used inside DeepDive, while varying the sampling time from 10 to $10k$ seconds. For each sampling algorithm, we measure the quality using the Kullback-Leibler (KL) divergence [27] between the estimated marginal probabilities using this algorithm and the true marginal probabilities provided by the ground truth. Figures 14(a) and 14(b) show the average KL divergence values for both sampling algorithms while building the GWDB and NYCCAS knowledge bases, respectively. Our proposed sampling achieves at least 49% and 41% less divergence values in the GWDB and NYCCAS cases, respectively, compared to the basic Gibbs sampling. This confirms the superiority of *Sya* in the inference quality results that have been shown in Figure 12(a).

VII. RELATED WORK

Traditional Knowledge Base Construction Systems. There is a wide array of knowledge base construction systems that are capable of extracting structured facts and relations. Such systems can be broadly categorized into two categories: *rule-based systems* (e.g., expert rules [12], [26] and crowdsourcing rules [4], [7]), and *machine learning-based systems* (e.g., classification [13], [15], maximum-a-posteriori models [25], [38], Markov Logic Networks (MLN) [9], [10], [36], and deep

learning [45]). We refer to these as "traditional" systems. The closest of these systems considering spatial attributes are [6] and [41], which augment facts with their location information (e.g., "lives at" attribute). However, no traditional system has exploited the location information between entities or facts during the construction. *Sya*, conversely, is the first MLN-based knowledge base construction system that considers such relationships to improve the knowledge base quality.

Geo-Knowledge Bases. Recent knowledge base systems have been proposed to extract facts about spatial entities (e.g., lakes) from Volunteered Geographic Information (VGI) [17] along with Semantic Geospatial Web [14] (see [5] for a comprehensive survey). In addition, a recent work has been focusing on the problem of entity alignment between knowledge bases with a special focus on spatial entities [40]. However, extracting and maintaining facts about spatial entities is a vastly different problem than we study in this paper. In *Sya*, we extract a knowledge base of generic facts, yet, we exploit the spatial information, if any, to improve the output quality.

Inference Techniques. The inference task uses a probabilistic inference algorithm to compute the factual score (i.e., probability) associated with generated relations. Existing inference algorithms in knowledge base construction systems are based on either Gibbs sampling [46], Markov chain Monte Carlo (MCMC) [1], [10], [28], [31], belief propagation [37], lifted inference [19], or specialized Markov Logic Network algorithms [22]. *Sya* provides a new variant of Gibbs sampling that adapts Concliques-based partitioning [23].

VIII. Conclusions

We introduced *Sya*, a full-fledged system that provides a native support for exploiting spatial relationships during the MLN-based knowledge base construction process. We introduced several extensions and optimization to provide the efficiency and scalability of the grounding and inference phases when dealing with spatially-correlated knowledge base relations. We also studied the trade-off between the inference quality and runtime of *Sya*. We also showed that *Sya* can significantly outperform the state-of-the-art MLN-based knowledge base construction systems in terms of accuracy and efficiency. In addition, *Sya* can be easily used to extend any of these systems to make it support spatial awareness.

References

[1] Alchemy. https://alchemy.cs.washington.edu/.

[2] L. Anselin, I. Syabri, and Y. Kho. GeoDa: An Introduction to Spatial Data Analysis. *Geographical Analysis*, 2006.

[3] W. G. Aref and H. Samet. Efficient Processing of Window Queries in the Pyramid Data Structure. In *PODS*, 1990.

[4] S. Auer et al. DBpedia: A Nucleus for a Web of Open Data. In *International Semantic Web Conference*, 2007.

[5] A. Ballatore, D. C. Wilson, and M. Bertolotto. A Survey of Volunteered Open Geo-Knowledge Bases in the Semantic Web. In *Quality Issues in the Management of Web Information*. Springer, 2013.

[6] J. Biega, E. Kuzey, and F. M. Suchanek. Inside YAGO2s: A Transparent Information Extraction Architecture. In *WWW*, 2013.

[7] K. Bollacker et al. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *SIGMOD*, 2008.

[8] R. Bose. Knowledge Management-enabled Health Care Management Systems. *Expert Systems with Applications*, 2003.

[9] Y. Chen and D. Z. Wang. Knowledge Expansion over Probabilistic Knowledge Bases. In *SIGMOD*, 2014.

[10] Y. Chen, X. Zhou, K. Li, and D. Z. Wang. Archimedes: Efficient Query Processing over Probabilistic Knowledge Bases. *SIGMOD Record*, 2017.

[11] CiteSeerX. http://citeseerx.ist.psu.edu/.

[12] O. Deshpande, D. Lamba, et al. Building, Maintaining, and Using Knowledge Bases: A Report from the Trenches. In *SIGMOD*, 2013.

[13] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang. From Data Fusion to Knowledge Fusion. In *PVLDB*, 2014.

[14] M. Egenhofer. Toward the Semantic Geospatial Web. In *SIGSPATIAL*, 2002.

[15] O. Etzioni et al. Open Information Extraction: The Second Generation. In *International Joint Conference on Artificial Intelligence*, 2011.

[16] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, 1987.

[17] M. F. Goodchild. Citizens as Sensors: The World of Volunteered Geography. *GeoJournal*, 2007.

[18] Google Knowledge Graph. https://www.google.com/intl/en-419/insidesearch/features/search/knowledge.html.

[19] E. Gribkoff and D. Suciu. SlimShot: In-database Probabilistic Inference for Knowledge Bases. In *PVLDB*, 2016.

[20] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. *SIGMOD Record*, 1984.

[21] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized Search Trees for Database Systems. In *VLDB*, 1995.

[22] S. Jiang, D. Lowd, and D. Dou. Learning to Refine an Automatically Extracted Knowledge Base Using Markov Logic. In *ICDM*, 2012.

[23] M. Kaiser, S. Lahiri, and D. Nordman. Goodness of Fit Tests for a Class of Markov Random Field Models. *The Annals of Statistics*, 2012.

[24] A. Kaplan. *On Advancing MCMC-based Methods for Markovian Data Structures with Applications to Deep Learning, Simulation, and Resampling*. PhD dissertation, Iowa State University, 2017.

[25] G. Kasneci, M. Ramanath, F. Suchanek, and G. Weikum. The YAGO-NAGA Approach to Knowledge Discovery. *SIGMOD Record*, 2009.

[26] R. Krishnamurthy et al. SystemT: A System for Declarative Information Extraction. *SIGMOD Record*, 2009.

[27] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 1951.

[28] K. Li et al. In-database Batch and Query-time Inference over Probabilistic Graphical Models using UDA-GIST. *VLDB Journal*, 2017.

[29] Y. Nahshon et al. Incorporating Information Extraction in the Relational Database Model. In *WebDB*, 2016.

[30] National Democratic Institute: Sanitation Levels. https://www.ndi.org/sites/default/files/WASH-WaterAid-Fact-Sheet.pdf, 2019.

[31] F. Niu, C. Ré, et al. Tuffy: Scaling Up Statistical Inference in Markov Logic Networks Using an RDBMS. In *PVLDB*, 2011.

[32] NYC OpenData. https://data.cityofnewyork.us/Environment/NYCCAS-Air-Pollution-Rasters/q68s-8qxv.

[33] Open Geospatial Consortium. http://www.opengeospatial.org/.

[34] M. Richardson and P. M. Domingos. Markov Logic Networks. *Machine Learning*, 2006.

[35] I. Sabek et al. A Demonstration of Sya: A Spatial Probabilistic Knowledge Base Construction System. In *SIGMOD*, 2018.

[36] J. Shin, S. Wu, F. Wang, C. D. Sa, C. Zhang, and C. Ré. Incremental Knowledge Base Construction Using DeepDive. In *PVLDB*, 2015.

[37] P. Singla and P. Domingos. Lifted First-order Belief Propagation. In *AAAI*, 2008.

[38] F. M. Suchanek, M. Sozio, and G. Weikum. SOFIE: A Self-organizing Framework for Information Extraction. In *WWW*, 2009.

[39] Texas Ground Water Database. www.twdb.texas.gov/groundwater/data/.

[40] B. D. Trisedya, J. Qi, and R. Zhang. Entity Alignment between Knowledge Graphs Using Attribute Embeddings. In *AAAI*, 2019.

[41] B. D. Trisedya, G. Weikum, J. Qi, and R. Zhang. Neural Relation Extraction for Knowledge Base Enrichment. In *ACL*, 2019.

[42] D. Venugopal and V. Gogate. On Lifting the Gibbs Sampling Algorithm. In *NIPS*, 2012.

[43] M. Wainwright and M. Jordan. Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends in ML*, 2008.

[44] World Health Organization: Liberia Ebola Data. http://apps.who.int/gho/data/node.ebola-sitrep.quick-downloads, 2019.

[45] S. Wu et al. Fonduer: Knowledge Base Construction from Richly Formatted Data. In *SIGMOD*, 2018.

[46] C. Zhang and C. Ré. Towards High-throughput Gibbs Sampling at Scale: A Study Across Storage Managers. In *SIGMOD*, 2013.

[47] C. Zhang and C. Ré. DimmWitted: A Study of Main-memory Statistical Analytics. In *PVLDB*, 2014.