Adopting Markov Logic Networks for Big Spatial Data and Applications

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Ibrahim Sabek

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Prof. Mohamed F. Mokbel

January, 2020

Acknowledgements

This thesis would not be possible without the kind help and support of many individuals around me. I would like to express my gratitude and sincere thanks to all of them.

First and foremost, I would like to thank my advisor, Prof. Mohamed F. Mokbel, for his continuous guidance and support of my research. He has been also an excellent role model for critical thinking, hard work, great passion, and commitment toward research. I would not have become the researcher that I am today without his invaluable lessons and assistance.

Besides my advisor, I will forever be indebted to my wife, Hebatalla, for her ongoing support, countless sacrifices as well as being my closest friend in pursuing our PhD career together. In addition, Hebatalla's family has always welcomed me as a son and brother. I am also a proud father of two amazing sons, Eyad and the expected little one, who are the joy of my life and made me more fulfilled than I could have ever imagined. I am also extremely thankful to my brother Amr, and my sisters Naglaa and Mai, for their unconditional faith and cheerleading regardless the remote distance between us.

I would like to thank the many former and current members of the DMLab that I was fortunate enough to be a part of. In particular, I am grateful to Ahmed Eldawy, Amr Magdy, Louai Alarabi, Rami Alghamdi, Saif Al-Harthi, Christopher Jonathan, Mashaal Musleh, and Harshada Chavan. I sincerely appreciate their help and contributions that greatly enriched my research and mentoring experience.

I also would like to extend my gratitude to Prof. Vipin Kumar, Prof. Shashi Shekhar, Prof. Arindam Banerjee, and Prof. Jarvis Haupt for serving on my thesis and preliminary examination committees. In particular, I am thankful to Prof. Shashi Shekhar for his constructive feedback and useful insights during the early stages of my PhD topic.

Last but not least, I would like to acknowledge the financial support from the National Science Foundation and University of Minnesota, including the assistantships from Department of Computer Science and Engineering and the Doctoral Dissertation Fellowship.

Dedication

To the memory of my mom, whose greatest wish for me was to obtain my PhD, to my wife and my sons, and finally, to all my family and friends.

Abstract

Markov Logic Networks (MLN) [1] have become a de-facto statistical learning and inference framework to perform efficient and user-friendly analysis on massive data, with many applications in knowledge base construction [2], data cleaning [3], information extraction [4], among others. Meanwhile, large-scale spatial data analysis has gained much interest in recent years due to the need for extracting insights from spatial data. However, analyzing spatial data using existing solutions typically cannot satisfy the scalability requirement of most applications as these solutions were not originally designed for the huge amounts of spatial data being generated at the moment (e.g., there are 10 Million geotagged tweets issued every day [5]). Unfortunately, none of these existing solutions exploits the power of the MLN framework to boost the usability, scalability, and accuracy of spatial analysis applications.

The main goal of this thesis is to provide the first research effort to combine the two worlds of MLN and spatial data analysis. We address the two main challenges that face any spatial analysis application when using MLN. The first challenge is how to modify the core processing and functionalities of MLN to make it aware with the distinguished features and needs of spatial data and applications. The core of MLN is composed of two main components, namely, *grounding* using factor graphs [6, 7] and *inference* using Gibbs sampling [8, 9]. The factor graph is used as the main data structure for learning and inferring the weights of the MLN features, while Gibbs sampling infers the values of model variables and computes their associated probabilities using the weighted MLN features. The second challenge is how to efficiently represent spatial analysis problems (e.g., spatial regression [10] and spatial probabilistic graphical modeling [11–13]) using MLN. This requires to find an equivalent first-order logic [14] representation for any input spatial analysis problem that makes sure that the input problem can be appropriately mapped and executed using the MLN framework.

This thesis makes the following contributions. First, we present *Sya*; the first spatial probabilistic knowledge base construction system based on the spatial-aware MLN framework. We show our spatial extensions to the different MLN layers, including *language*, *grounding* and *inference*, implemented inside a knowledge base construction application. We then introduce three scalable spatial analysis systems, namely, *TurboReg*, *RegRocket*, and *Flash*, that are equipped with efficient first-order logic representations for different spatial analysis problems.

In *TurboReg*, we provide a scalable framework for employing binary autologistic models in predicting large-scale spatial phenomena. In *RegRocket*, we provide an efficient extension for *TurboReg* to support the multinomial (i.e., categorical) autologistic models. Finally, in *Flash*, we show how we can scale up the performance of spatial probabilistic graphical modeling, which is an important class of spatial data analysis, using MLN. In summary, the research contributions in this thesis have advanced the state-of-the-art in scalable spatial analysis, by presenting efficient MLN-based algorithms with strong theoretical foundations.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Era of Big Spatial Data and Applications

Recently, there has been a proliferation in the amounts of spatial data produced from several devices such as smart phones, space telescopes, and medical devices. For example, space telescopes generate up to 150 GB weekly spatial data [15], medical devices produce spatial images (X-rays) at a rate of 50 PB per year [16], a NASA archive of satellite earth images has more than 500 TB and is increasing daily by around 25 GB [17], while there are 10 Million geotagged tweets issued from Twitter every day as 2% of the whole Twitter firehose [18, 19].

Various applications and agencies need to analyze these unprecedented amounts of spatial data [20]. For example, the Blue Brain Project [21] studies the brain's architectural and functional principles through modeling brain neurons as spatial data [22]. Epidemiologists use spatial analysis techniques to identify cancer clusters [23], track infectious disease [24], and drug addiction [25]. Meteorologists study and simulate climate data through spatial analysis [26–29]. News reporters use geotagged tweets for event detection and analysis [30]. Such spatial applications raise the ongoing need for efficient spatial-aware data analysis and machine learning frameworks that allow data scientists and developers to analyze and turn the massive spatial data into useful insights.

1.2 Markov Logic Networks

The last two decades have witnessed significant efforts from researchers and practitioners worldwide to develop numerous machine learning and artificial intelligent methods, e.g., deep learning [31–34]. However, the expertise skills and efforts needed to deploy these methods become a major blocking factor in having a wide deployment of machine learning and artificial intelligence applications. As a result, Markov Logic Networks (MLN) [1, 35] were recently introduced to reduce this gap. In particular, the MLN framework combines first-order logic [14] with probabilistic graphical models to efficiently represent statistical learning and inference problems with few logical rules (e.g., rules with imply and bit-wise AND predicates). With MLN, data scientists and developers do not need to worry about the underlying machine learning work. Instead, they will focus their efforts on developing the rules that represent their applications. A machine learning application that may take thousands of lines of code can be expressed with just few MLN logic rules. This had a significant impact on the wide deployment of machine learning techniques in various applications, including knowledge base construction [2, 36, 37], data cleaning [3], information extraction [4], entity resolution [38], natural language processing [39], genetic analysis [40, 41], among others.

Unfortunately, researchers never take advantage of the recent advances of Markov Logic Networks (MLN) to boost the usability, scalability, and accuracy of spatial machine learning tasks (e.g., spatial regression [10]) used in the spatial applications. Furthermore, MLN-based applications (e.g., knowledge base construction [2]) would miss important results and have less accuracy when dealing with spatial data. The main reason is that the MLN framework is oblivious to the spatial data. The only way to support spatial data in MLN is to simply ignore its distinguished properties (e.g., spatial relationships among objects) and deal with it as non-spatial data. While this would work to some extent, it will result in a sub-par performance.

1.3 Thesis Contributions

The goal of this thesis is to adopt Markov Logic Networks (MLN) for big spatial data and applications [42]. To that end, we introduce two orthogonal, but related, research contributions. The first contribution is injecting the spatial awareness inside the MLN-based techniques and applications (e.g., knowledge base construction), which will result in a higher accuracy for such applications. The second contribution is taking advantage of the recent advances in MLN to boost the usability, deployment, scalability, and accuracy of long lasting spatial data analysis techniques. These two research contributions are briefly described in the rest of this section.

1.3.1 Spatial-aware MLN-based Knowledge Base Construction

Knowledge base construction has been an active area of research over the last two decades with several system prototypes coming form academia and industry, along with vital applications. Most recently, knowledge base construction systems employed the idea of Markov Logic Networks (MLN) to associate each extracted relation with a probability of how confident is the system that this relation is factual. These systems usually employ two main phases; an MLN-based grounding phase, which is responsible on how the factual scores correlate with each others, and an MLN-based inference phase, which is responsible on calculating the final probability of each extracted relation. DeepDive [2,9,43], an MLN-based system, has emerged as one of the most popular probabilistic knowledge base construction systems [2, 9, 43], applied in vital applications like human trafficking [44,45], geology [46], and paleontology [47]. Unfortunately, probabilistic knowledge base systems do not fully utilize the underlying spatial information, which results in less accuracy in the factual scores.

In this contribution, we provide a native support for spatial data within Markov Logic Networks (MLN), and exploit such support to build a spatial-aware probabilistic knowledge base construction system, called *Sya* [59, 60]. First, *Sya* pushes the spatial awareness inside the internal data structures and core learning and inference functionalities of MLN. In particular, *Sya* implements an efficient spatial variation of the three main modules of MLN, namely, *language*, *grounding*, and *inference*. Then, *Sya* extends DeepDive [2, 9, 43] with the proposed spatial-aware MLN components to generate more accurate knowledge base outputs in case of having spatial information. Our experimental results, through building two real knowledge bases, have shown that *Sya* can achieve 70% higher F1-score on average over DeepDive, while achieving at least 20% reduction in the execution times.

1.3.2 MLN-based Spatial Data Analysis

Same as Markov Logic Networks (MLN) made it possible for data scientists and developers to embrace the difficulty of deploying machine learning techniques, we use our spatial-aware

MLN as a backbone infrastructure to support long lasting spatial analysis techniques that lack scalability as well as suffer from difficulty of deployment. Specifically, we address the following two main spatial analysis operations:

- The first operation is autologistic regression, which has been used in various applications to predict future values from previous data sets [10, 48–54]. To scale up the performance of autologistic regression, we introduce *TurboReg* [61], a framework that exploits the MLN framework to learn the autologistic regression parameters in an accurate and efficient manner. *TurboReg* provides an equivalent first-order logic representation to the dependency relations among neighbors in autologistic models, a long with a theoretical foundation for such representation. Once this equivalent representation is obtained, the autologistic model parameters can be easily learned through the powerful MLN framework. Our extensive experiments have shown that *TurboReg* achieves at least three orders of magnitude performance gain over the best state-of-the-art existing method while preserving the model accuracy. We further introduce *RegRocket* [62], an efficient extension for *TurboReg*, along with a theoretical foundation, to support the categorical (i.e., multinomial) prediction and predictor variables.
- The second operation is spatial probabilistic graphical modeling, which is an important class of spatial data analysis that provides efficient probabilistic graphical models for spatial data [11–13, 55]. This operation has been extensively used in many spatial and spatio-temporal applications including meteorology [56], flood risk analysis [57], and environmental science [58]. To scale up the performance of spatial probabilistic graphical modeling, we introduce *Flash* [63, 64]; a framework that goes beyond the autologistic regression problem and provides a more generic framework for scalable spatial data analysis based on MLN. To show the effectiveness of *Flash*, we provide MLN-based representations for three popular spatial probabilistic graphical models, namely, spatial Markov random fields [11], spatial hidden Markov models [12], and spatial Bayesian networks [13, 55]. We have evaluated *Flash*, based on three real spatial analysis applications, and achieved at least two orders of magnitude speed up in learning the modeling parameters over state-of-the-art computational methods.

1.4 Thesis Outline

The rest of this thesis focuses on addressing the abovementioned two research contributions. We summarize the thesis organization as follows:

- Chapter 2 gives a background on the Markov Logic Networks (MLN) framework. It describes the details of two main MLN components; first-order logic rules and factor graph representation. It also discusses how to represent any learning or inference problem using MLN and what are the main assumptions to have a valid representation.

- Chapter 3 introduces *Sya* [59, 60]; the first spatial probabilistic knowledge base construction system based on MLN.

- Chapter 4 introduces *TurboReg* [61]; an MLN-based scalable framework for using binary autologistic regression models in predicting large-scale spatial phenomena.

- Chapter 5 introduces *RegRocket* [62]; the first scalable framework for building autologistic models with multinomial prediction and predictor variables based on MLN.

- Chapter 6 introduces *Flash* [63, 64]; a framework that exploits MLN to scale up the performance of spatial probabilistic graphical modeling.

- Chapter 7 highlights the key contributions of our work and concludes the thesis.

Chapter 2

Preliminaries of Markov Logic Networks

Markov Logic Networks (MLN) have recently emerged as a powerful framework to efficiently learn and infer the parameters of data models with complex dependencies and distributions [35, 65, 66]. The MLN framework combines probabilistic graphical models (e.g., factor graphs [7]) with first-order logic [14] to perform probabilistic learning and inference based on logic constraints, where logic handles model complexities and probability handles uncertainty. MLN have been successfully applied in a wide span of data intensive applications including knowledge bases construction (e.g., DeepDive [2, 9, 43, 46], ProbKB [36, 67, 68], Archimedes [37, 69], Others [4, 65]), machine learning models (e.g., classification [70], data cleaning [3], ontology matching [71], entity resolution [38], natural language processing [39]), and genetic analysis [40, 41]. The success stories in such applications motivate us to explore MLN in computing the parameters of models with spatial dependencies such as spatial-aware knowledge bases (Chapter 3), autologistic regression (Chapter 4 and 5), and spatial probabilistic graphical modeling (Chapter 6). The rest of this chapter provides the details of main MLN components.

2.1 First-order Logic Rules

First-order logic (FOL) predicates [14] (e.g., conjunction, disjunction and implication) are used to encapsulate the knowledge in MLN. Any FOL formula consists of one or more FOL predicates. For example, causality relationships in MLN are usually represented using the following

FOL formula: $P_1 \wedge ... \wedge P_n \implies A$, where predicates $P_1 \wedge ... \wedge P_n$ together define an action A. FOL allows users to define universal or existential quantifiers in their formulas. For example, the following two rules represent two FOL formulas:

$$(\forall x) LowSanitation(x) \implies Ebola(x) \tag{2.1}$$

$$(\forall x) Ebola(x) \wedge Close(x, y) \implies Ebola(y) \tag{2.2}$$

where the first formula states that if there is a low sanitation level at location x then this location has a high risk of Ebola infection, and the second formula states that if the location x has a high risk of Ebola infection and it is close to another location y then there is a high risk of Ebola infection at location y as well.

The MLN framework employs FOL formulas to represent the model constraints. In particular, they adapt a weighted variation of FOL formulas, where each formula is associated with weight, whether this weight is constant or needs to be learned from the training data. For the formulas in the above example, the weighted variation used in MLN is as follows:

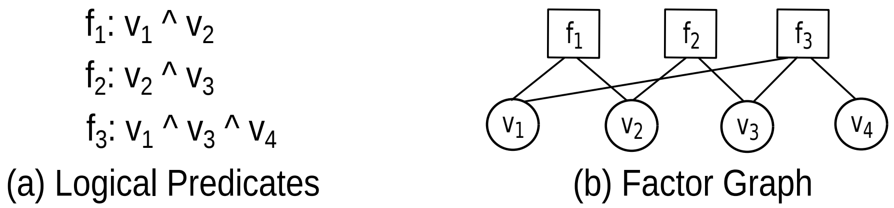
$$w_1 \qquad (\forall x) LowSanitation(x) \implies Ebola(x) \tag{2.3}$$

$$w_2 \qquad (\forall x) Ebola(x) \wedge Close(x, y) \implies Ebola(y) \tag{2.4}$$

The weights associated with FOL formulas indicate the confidence in the knowledge represented by these formulas. The value of any weight can be any positive value as long as larger values are assigned to formulas with high confidence. In some implementations, negative values are used to indicate the negation of the formula. Many efficient logic programming frameworks have been proposed to generate first-order logic predicates on a large-scale such as DDlog [2], XLog [72], and fuzzy logic [73].

2.2 Factor Graph Representation

Any MLN model consists of a set of weighted FOL formulas (e.g., Formula 2.3 and 2.4). To learn the values of weights $\mathcal{W} = \{w_1, ..., w_h\}$ associated with predicates in these formulas,



(a) Logical Predicates (b) Factor Graph

Figure 2.1: Translating First-order Logic Predicates into A Factor Graph in MLN.

these MLN predicates are translated into an equivalent probabilistic graphical model, namely factor graph [7], which has $\mathcal{W}$ as the parameters of its joint probability distribution. By doing that, the problem of learning $\mathcal{W}$ is reduced into the problem of learning the joint distribution of this factor graph. In general, the factor graph $\mathcal{G}$ in MLN consists of the following:

- Variable nodes $\mathcal{V}$ which are groundings of the predicates in the FOL formulas based on a set of constants $C = \{c_1, c_2, ..., c_n\}$ (i.e., different constants are used to generate different instances of the same formula). Each variable node $v \in \mathcal{V}$ has a binary value dependent on the value of the grounding. As the number of constants and formula groundings increases, the MLN model grows, but the number of the FOL formulas stays the same. The following examples show two groundings (i.e., instantiations) for Formula 2.3 when having two cities (i.e., constants) from Liberia (i.e., $x \in \{Margibi, Bong\}$).

$$w_1 \qquad LowSanitation(Margibi) \implies Ebola(Margibi) \tag{2.5}$$

$$w_1 \qquad LowSanitation(Bong) \implies Ebola(Bong) \tag{2.6}$$

- Factor nodes $\mathcal{F}$ which connect the variable nodes $\mathcal{V}$ whose corresponding predicates appear in the same MLN formula. There is an edge between any factor node f and each variable node v that is connected to f.

- Each factor node $f \in \mathcal{F}$ is associated with a feature. It is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the weight of the formula.

Figure 2.1 shows an example of translating three bitwise-AND logical predicates defined over an example of MLN that has four variables (v_1, v_2, v_3 and v_4) into a factor graph.

Probability Distribution. The full joint distribution of variables $\mathcal{V}$ in a factor graph $\mathcal{G}$ can be estimated in terms of the factors $\mathcal{F}$ and their weights $\mathcal{W}$ as a log-linear model [35]:

$$Pr(\mathcal{V} = v) = \frac{1}{Cons} \exp\Big(\sum_{i=1}^{h} w_i f_i(v)\Big) \tag{2.7}$$

where $Cons$ is a normalization constant, $f_i(v)$ is the value of whether the i-th formula is satisfied or not, and w_i is its weight. Scalable optimization techniques have been proposed to efficiently learn the values of weights $\mathcal{W}$ in factor graph, such as gradient descent optimization [8, 74, 75]. In such techniques, the joint distribution of variables $\mathcal{V}$ is usually estimated through the conditional probability distribution $Pr(\mathcal{V}|\mathcal{E})$ where $\mathcal{E}$ is a set of known evidence variables and $\mathcal{V}$ represents the variables whose values are to be determined only.

2.3 Modeling with Markov Logic Networks

Any model can be represented with MLN, only if it has two main properties: (1) the model can be represented as a set of p binary random variables (corresponding to the MLN variables $\mathcal{V} = \{v_1, ..., v_p\}$ ($v_i \in \{0, 1\}$)), and (2) the dependencies between model variables can be described with a set of weighted constraints defined over them (corresponding to the weighted MLN factors $\mathcal{F} = \{f_1, ..., f_h\}$), where the weights of these constraints are the model parameters that need to be learned. The constraints describe how the values of the random variables correlate with each other. A model with these two properties can exploit MLN to learn weights $\mathcal{W}$ that maximize the probability of satisfying model constraints $\mathcal{F}$.

Example. Assume a model of two variables v_{prof} and v_{teach}, where v_{prof} denotes whether a person is professor or not, and v_{teach} denotes whether a person teaches or not. We can define a constraint that ”if a person is a professor then she teaches, and vice versa”. This constraint can be represented as a bitwise-AND predicate $v_{prof} \wedge v_{teach}$. In this case, the MLN framework learns a weight w that maximizes the probability of v_{prof} and v_{teach} having the same value (i.e., either $v_{prof} = 1$ and $v_{teach} = 1$ or $v_{prof} = 0$ and $v_{teach} = 0$).

Chapter 3

Sya: Enabling Spatial Awareness inside Probabilistic Knowledge Base Construction

3.1 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., [76, 77]) and industry (e.g., [78–80]), along with many important applications, e.g., web search [81], digital libraries [82], and health care [83]. 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 [2, 36, 37, 84]). An example of such probabilistic relations is "Alice is a spouse of Bob with 80% probability".

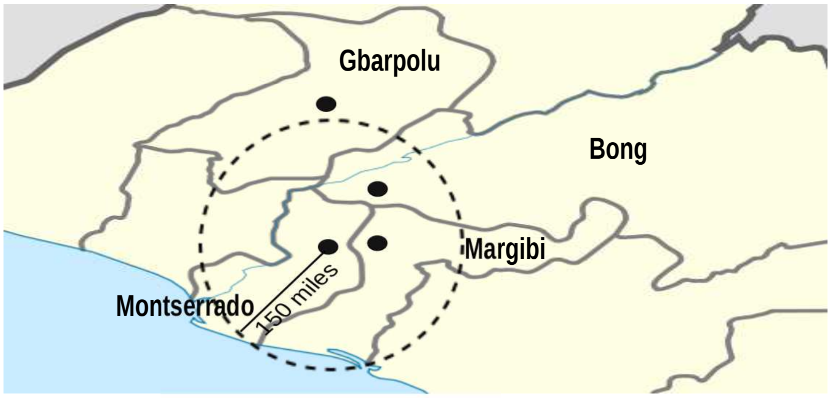
Recently, Markov Logic Networks (MLN) [1] have been a standard tool for building probabilistic knowledge base construction systems. Examples of such systems include DeepDive [2], ProbKB [36], and Archimedes [37]. In these MLN-based systems, users express the knowledge base construction logic using a set of first-order logic rules [14]. Then, such rules are processed

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

Figure 3.1: Factual Scores of EbolaKB Using DeepDive and *Sya*.

on two steps: 1) *grounding*, which evaluates the rules to construct a ground factor graph [6] 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.

Example. We used DeepDive [2], 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 [85] in various counties in Liberia, namely EbolaKB. Figure 3.1(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:

```
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 3.1(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 [86] (first column in Figure 3.1(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 3.1(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 3.1(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 factual 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 3.6, this comes with tremendous latency in the grounding step which makes it impractical to build knowledge bases.

Approach. In this chapter, 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 [6] 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 [6], 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 [8] 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 [87]. Thus, a new efficient variation of Gibbs sampling is needed to handle these spatial correlations.

Contributions. Our technical contributions in this chapter 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 3.2).
- We extend a popular datalog-like language, namely DDlog [2], with spatial constructs that allow users to easily express their spatial semantics (Section 3.3).
- We introduce a new spatial variation of the factor graph [6], 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 3.4).
- We introduce a new variant of Gibbs Sampling, namely *Spatial Gibbs Sampling*, that exploits the Conclique [88] 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 [87] (Section 3.5).
- We perform an extensive evaluation of *Sya* with DeepDive [2] through building two real knowledge bases about the water quality in Texas [89], namely GWDB, and the air pollution in New York city [90], 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 3.6).

3.2 Sya Architecture

Figure 3.2 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

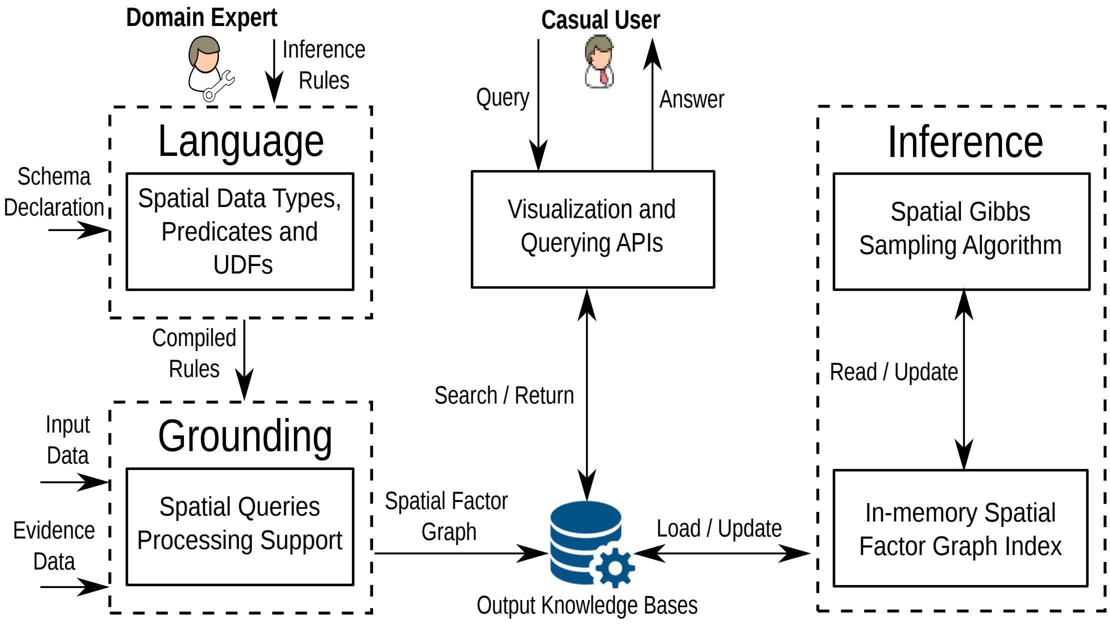
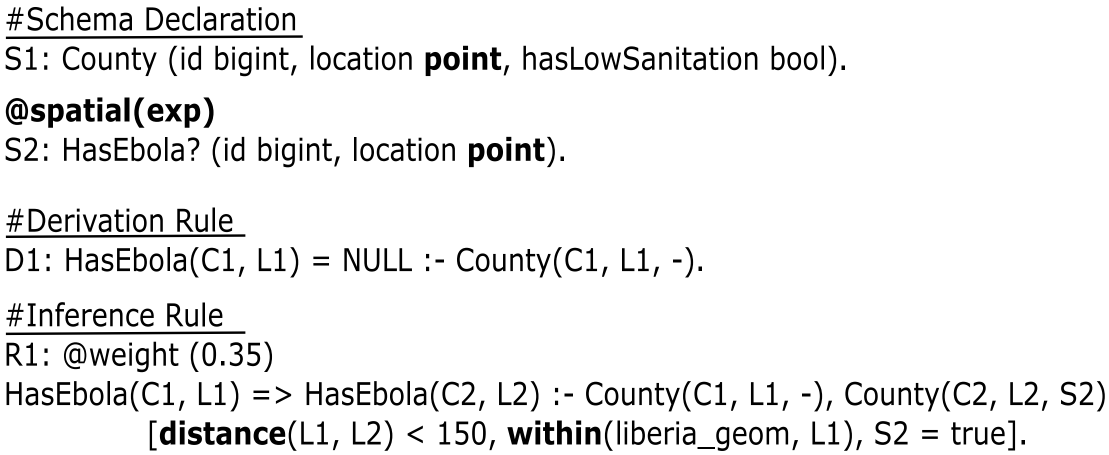


Figure 3.2: *Sya* System Architecture.

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 [2], 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) the 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 3.3.

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 [6] 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 3.4.



```
#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].
```

Figure 3.3: 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 [91], referred to as *In-memory Spatial Factor Graph Index*, that partitions the spatial factor graph variables and correlations into a set of concliques [88], 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 3.5.

3.3 The Language Module

Users of MLN-based knowledge base construction systems can use either native first-order clauses [14] (e.g., as in ProbKB [36], and Archimedes [37]) or high-level datalog-like languages (e.g., as in DeepDive [2] and SpannerLog [92]) 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 3.4).

In *Sya*, instead of providing a completely new language, we choose to extend the DDlog [2] 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 spatial extensions conform to the Open Geospatial Consortium (OGC) standard [93].

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 the random variables that will be used later. 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 the input relation $Data(s)$. 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 3.3, which builds the EbolaKB knowledge base in Section 3.1.

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 3.3, 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., the *HasEbola* relation in Statement $S2$ in Figure 3.3). The `w` input in `@spatial(w)` specifies the type of spatial weighing function used during the grounding and inference steps (details are in Section 3.4 and 3.5, respectively). 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 [94] function that is already implemented in *Sya*.

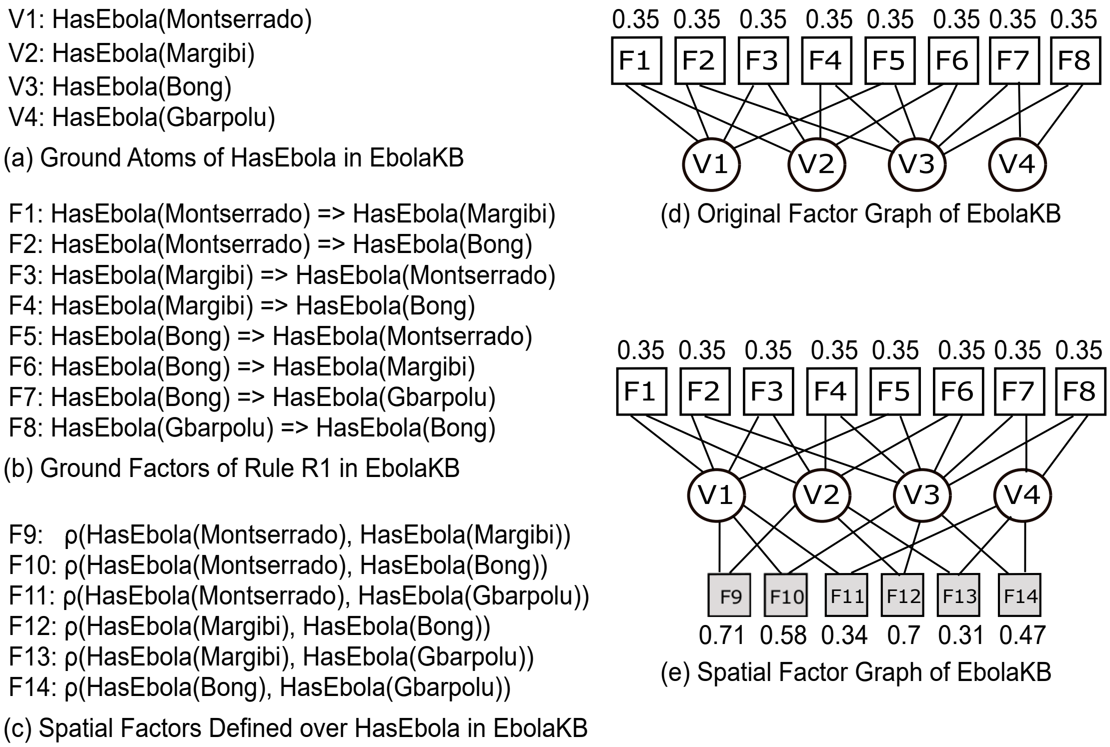
Spatial Predicates. *Sya* extends the body of DDlog rules with spatial predicates (e.g., `overlaps`, `within`, and `distance`) and functions (e.g., `union` and `buffer`) to support the evaluation of spatial queries in the grounding module (details are in Section 3.4). Spatial predicates can be composed. For example, the inference rule $R1$ in Figure 3.3, 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, the automatic extraction of 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].

3.4 The Grounding Module

The knowledge base construction rules represented by either native first-order clauses or datalog-like languages (as shown in Section 3.3) 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

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



V1: HasEbola(Montserrado)
V2: HasEbola(Margibi)
V3: HasEbola(Bong)
V4: HasEbola(Gbarpolu)

(a) Ground Atoms of HasEbola in EbolaKB

F1: HasEbola(Montserrado) => HasEbola(Margibi)
F2: HasEbola(Montserrado) => HasEbola(Bong)
F3: HasEbola(Margibi) => HasEbola(Montserrado)
F4: HasEbola(Margibi) => HasEbola(Bong)
F5: HasEbola(Bong) => HasEbola(Montserrado)
F6: HasEbola(Bong) => HasEbola(Margibi)
F7: HasEbola(Bong) => HasEbola(Gbarpolu)
F8: HasEbola(Gbarpolu) => HasEbola(Bong)

(b) Ground Factors of Rule R1 in EbolaKB

F9: ρ(HasEbola(Montserrado), HasEbola(Margibi))
F10: ρ(HasEbola(Montserrado), HasEbola(Bong))
F11: ρ(HasEbola(Montserrado), HasEbola(Gbarpolu))
F12: ρ(HasEbola(Margibi), HasEbola(Bong))
F13: ρ(HasEbola(Margibi), HasEbola(Gbarpolu))
F14: ρ(HasEbola(Bong), HasEbola(Gbarpolu))

(c) Spatial Factors Defined over HasEbola in EbolaKB

(d) Original Factor Graph of EbolaKB

(e) Spatial Factor Graph of EbolaKB

Figure 3.4: Example on *Sya* Grounding for EbolaKB.

structure, called factor graph [6]. 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 3.3). The generated random variables are called *ground atoms*. Figure 3.4(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 3.3) that satisfies the predicates and conditions in the body of this rule. The generated factors are called *ground factors*. Figure 3.4(b) shows an example of ground factors of rule $R1$ in the EbolaKB example that satisfy the `distance` and `within` predicates. Figure 3.4(d) depicts an example ground factor graph based on ground atoms and factors from figures 3.4(a) and 3.4(b), respectively. Each factor is represented by a square, and has edges with its variables 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:

$$P(\mathcal{V} = v) = \frac{1}{Z} \prod_{f_i \in \mathcal{F}} f_i(\mathcal{V}_i) = \frac{1}{Z} \exp\Big(\sum_{f_i \in \mathcal{F}} w_{f_i} n_{f_i}(v) \Big) \tag{3.1}$$

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 3.1 represents the marginal inference, which is commonly used in the literature of probabilistic knowledge bases. Another type of inference is maximum a posteriori (MAP), in which we find the most likely variables assignment, is out of scope of this work.

In this section, we describe how *Sya* extends the ground factor graph to support spatially-correlated ground atoms (Section 3.4.1). In addition, we discuss the database support in *Sya* for constructing the factor graph in an efficient manner (Section 3.4.2). Finally, we provide an optimization to prevent the combinatorial explosion that could happen during the grounding of spatial factor graph (Section 3.4.3).

3.4.1 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" (a.k.a first rule of geography [95]). 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 3.3). 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 to be 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 (Definition 1), then we extend this definition for the case of *categorical* variables (Definition 2).

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*

$$\rho_{j,k} = \begin{cases} e^{w_{d(v_j,v_k)}} & v_j = v_k \\ e^{-w_{d(v_j,v_k)}} & otherwise \end{cases} \tag{3.2}$$

As shown in Equation 3.2, 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 speaking, spatial correlations can be defined on more than two grounds. However, we focus only on binary correlations in *Sya*. The extension to high-order cases is intuitive as well, but, out of scope of this work.

We propose the spatial factor $\rho_{j,k}$ in an exponential form to easily extend the probability distribution $P(\mathcal{V} = v)$ in Equation 3.1 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:

$$\begin{aligned} P(\mathcal{V} = v) = \frac{1}{Z} \exp \Big(& \sum_{f_i \in \mathcal{F}} w_{f_i} n_{f_i}(v) \\ & + \sum_{\rho_{j,k} \in \rho} w_{d(v_j, v_k)} \big(\mathbb{1}_{v_j = v_k} - \mathbb{1}_{v_j \neq v_k} \big) \Big) \end{aligned} \tag{3.3}$$

where $\mathbb{1}_{v_j=v_k}$ and $\mathbb{1}_{v_j \neq v_k}$ are indicator functions. Figure 3.4(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 the 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 3.3 indicates that $w_{d(v_j,v_k)}$ should be calculated using the exponential distance weighing [94] function. Figure 3.4(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 [2]. As a result, we adapt the spatial factor function in Equation 3.2 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*

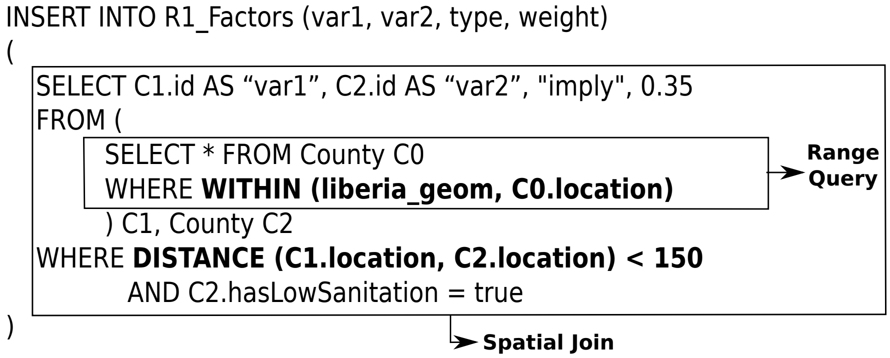
$$\rho_{j,k}(t_j, t_k) = \begin{cases} e^{w_{d(v_j,v_k)}} & v_j(t_j) = v_k(t_k) = 1, t_j = t_k \\ e^{-w_{d(v_j,v_k)}} & v_j(t_j) = v_k(t_k) = 1, t_j \neq t_k \\ 1 & otherwise \end{cases} \tag{3.4}$$

Similar to Equation 3.2, Equation 3.4 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 3.3. 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 3.4.3.

3.4.2 Rules Translation and Execution

Existing MLN-based knowledge base construction systems (e.g., [2, 36]) efficiently construct the factor graph by evaluating its corresponding inference rules as SQL queries to exploit the scalability and efficiency of DBMS execution engines. 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 3.3) 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 [96] and GIST [97]) 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 3.5 shows an example of translating the inference rule $R1$ from Figure 3.3, which has two spatial predicates `distance` and `within` that are translated into a spatial join and range



```
INSERT INTO R1_Factors (var1, var2, type, weight)
(
 SELECT C1.id AS “var1”, C2.id AS “var2”, "imply", 0.35
 FROM (
      SELECT * FROM County C0
      WHERE WITHIN (liberia_geom, C0.location)
      ) C1, County C2
 WHERE DISTANCE (C1.location, C2.location) < 150
        AND C2.hasLowSanitation = true
)
```

Figure 3.5: Example on Rules Translation in *Sya*.

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.

Integration with Spatial Databases. *Sya* fully integrates with scalable spatial database engines, e.g., PostGIS, and MySQL Spatial, to execute the translated queries in the grounding module. 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*.

3.4.3 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 3.4.1). 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

Algorithm 1 Function ActiveSpatialFactors (SpatialVariables $\mathcal{V}_s$, Domains $\mathcal{D}_s$, SpatialFactors ρ, Threshold T)

```
$Active \leftarrow Null$
for each $v \in \mathcal{V}_s$ do
    for each $(i, j) \in \mathcal{D}_s(v) \times \mathcal{D}_s(v)$ do
        if $(Pr(i|j) \geq T)$ & $(Pr(j|i) \geq T)$ then
            for each $k \in \rho(v)$ do
                $Active \leftarrow Active \cup k(i, j)$
            end for
        end if
    end for
end for
return $Active$
```

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 is application-dependent, and 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 3.6. Algorithm 1 depicts the pseudo code for the proposed optimization.

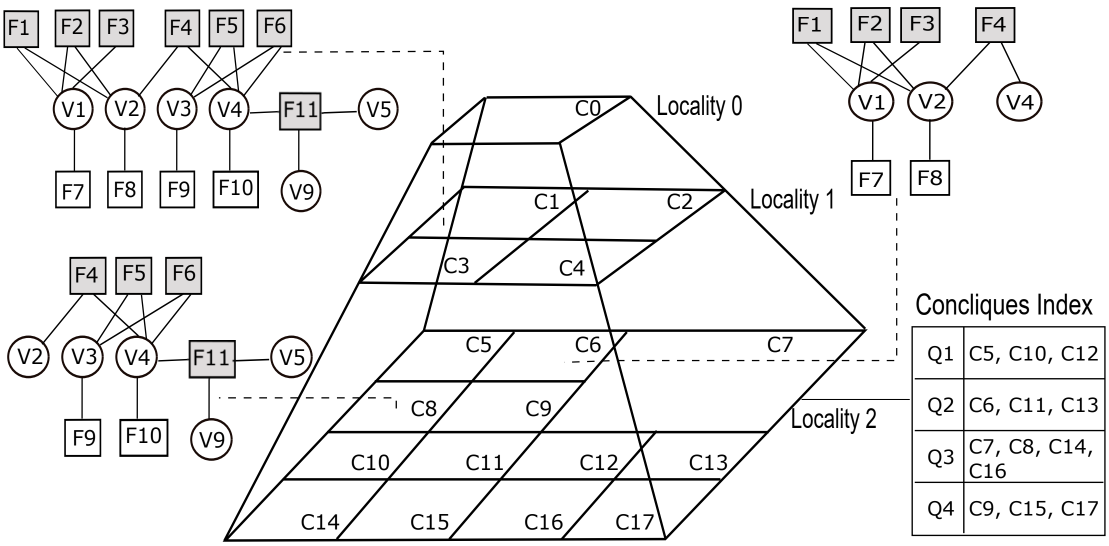
3.5 The Inference Module

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 perform this step in MLN-based knowledge base construction systems, approximate inference via Gibbs sampling is commonly used [8, 37, 98, 99]. 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 [88]. In this section, we provide a new Gibbs sampling algorithm, namely *Spatial Gibbs Sampling*, that overcomes this limitation by employing a new combination of 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 [8, 99] 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., [100]). 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 [91], with a well-known spatial statistics concept, namely concliques [88], 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 [87]. First, we give the details of the pyramid index and concliques concepts used in *Sya*. 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 [91] 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.



Q1	C5, C10, C12
Q2	C6, C11, C13
Q3	C7, C8, C14, C16
Q4	C9, C15, C17

Figure 3.6: Example on In-memory Pyramid Index of Spatial Factor Graph.

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 3.6 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 cell C_1 is divided into two sub graphs at cells C_6 and C_8 because they are children of cell C_1. Also, factor node F_4 is replicated in both cells C_6 and C_8 because it is connected to variables 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

Algorithm 2 Function SpatialGibbsSampling (SpatialFactorGraph $\mathcal{G}$, Instances K, Epochs E)

```
$Ctr \leftarrow$ *Null* /* Sampling Counters */
for all $v \in \mathcal{V}$ do in parallel
    $Ctr[v] \leftarrow 0$
$e \leftarrow \frac{E}{K}$ /* No. of Epochs Per Instance*/
$P \leftarrow$ BuildPyramidIndexOfSpatialFactorGraph ($G$)
$Q \leftarrow$ BuildConcliquesOfPyramidIndex ($P$)
$L \leftarrow$ No. of Levels in $P$
while $e \neq 0$ do
    for all $k \in \{1, 2, ..., K\}$ do in parallel
        for all $l \in \{2, 3, ..., L-1\}$ do serially
            $T \leftarrow$ GetNonEmptyCells ($P$, $l$)
            $U \leftarrow$ GetMinConcliquesCover ($Q$, $l$, $T$)
            for all $u \in U$ do serially
                for all $t \in T \cap u$ do in parallel
                    $Ctr_k[\mathcal{V}_t] \leftarrow$ RunStandardGibbsSampler ($\mathcal{V}_t$, $\mathcal{G}$, $Ctr_k$)
    $Ctr \leftarrow \frac{\sum_{k=1}^{K} Ctr_k}{K}, e--$
end while
for all $v \in \mathcal{V}$ do in parallel
    $v.Prob \leftarrow$ CalcMarginalProbability ($Ctr$, $v$)
```

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 [88]. For example, the cells of locality level 2 in Figure 3.6 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 3.6. 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. Algorithm 2 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 Ctr, 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 3.6 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 [2] as it is computationally-efficient, easy-to-implement, and can support incremental statistical 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 inference results as shown in block-based Gibbs sampling algorithms [100]. 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 Ctr 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 (Lines 8 to 17), and then use the final counts of samples to calculate and update the marginal probability of each variable as in [6](Lines 18 and 19).

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

Table 3.1: Statistics of KBs Used in Experiments.

Complexity. The complexity of Algorithm 2 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 $O((\frac{E}{K})|\mathcal{V}|^2)$.

3.6 Experiments

In this section, we experimentally evaluate the quality and scalability of *Sya*, based on a real system implementation [60] inside DeepDive [2]. 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.

3.6.1 Experimental Setup

Datasets. In our experiments, we have built two knowledge base systems, namely GWDB and NYCCAS, using both *Sya* and DeepDive. Table 3.1 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 details of these systems are as follows:

- 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 [89], which is collected by Texas Water Development Board (TWDB) about 9831 water wells. It contains

[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].
```

Figure 3.7: Example on a Rule for the GWDB KB in *Sya* and DeepDive.

information about each well such as location, depth and the concentration of different elements such as 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) [90] 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 3.7 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:

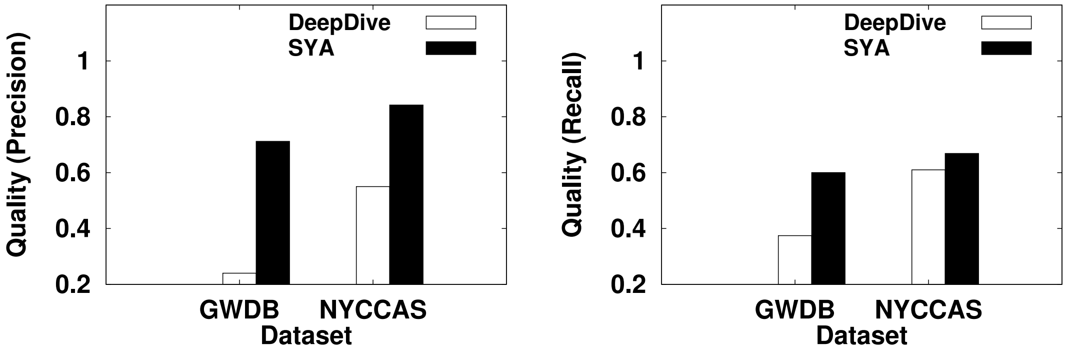
- *Precision* ($Prec$): the number of predicted factual scores that match the ground truth within 0.1 error (i.e., correctly inferred factual scores), over the total number of factual scores that will be predicted.

- *Recall* (Rec): the number of correctly inferred factual scores (calculated similar to $Prec$), over the total number of factual scores that should be predicated correctly according to the input evidence data.

- *F1-score*: the harmonic mean of precision and recall, which is calculated as:

$$F1 = \frac{2(Prec * Rec)}{Prec + Recall}$$

Environment. Both *Sya* and DeepDive 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 the PostgreSQL DBMS [101], and its spatial extension PostGIS [102], to execute SQL queries.



(a) Dataset vs. Precision

(b) Dataset vs. Recall

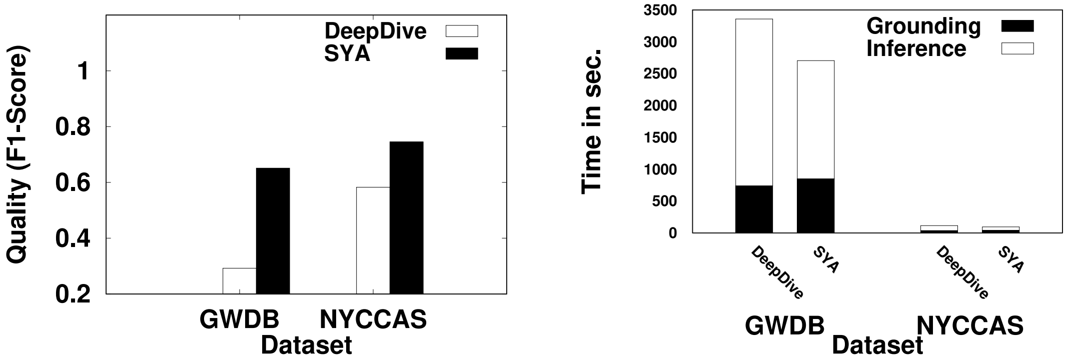
Figure 3.8: Comparison of *Sya* with DeepDive (Precision and Recall).

Parameters. Unless otherwise mentioned, we set the number of inference epochs to 1000, the input of the `@spatial` parameter (Section 3.3) to the exponential distance weighing function [94], 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).

3.6.2 Experimental Results

Comparison with DeepDive using Different Datasets

Figure 3.8(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 our work). 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



(a) Dataset vs. F1-Score

(b) Dataset vs. Exec. Time

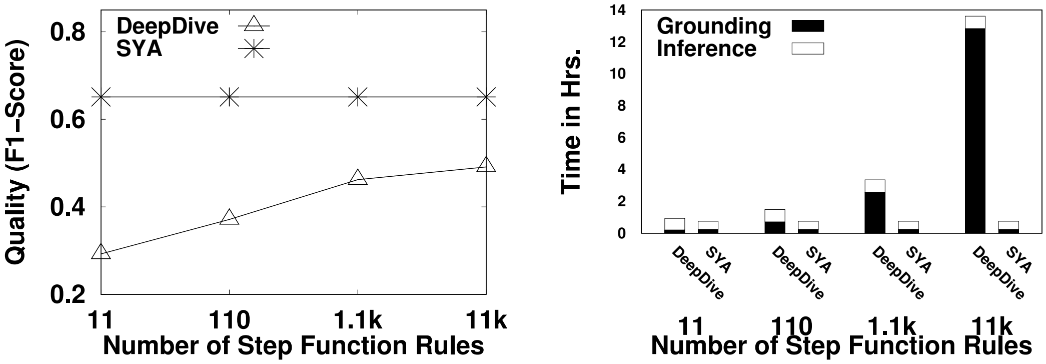
Figure 3.9: Comparison of *Sya* with DeepDive (F1-Score and Execution Time).

spatial predicates as a boolean 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 3.8(b) shows the recall results obtained by *Sya* and DeepDive while building the GWDB and NYCCAS knowledge bases. For the GWDB knowledge base, we still have the same conclusion that *Sya* is better than DeepDive. In this case, the improvement ratio is around 60%. For the NYCCAS knowledge base, 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 3.9(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 outputs.

Figure 3.9(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



(a) No. of Rules vs. F1-Score

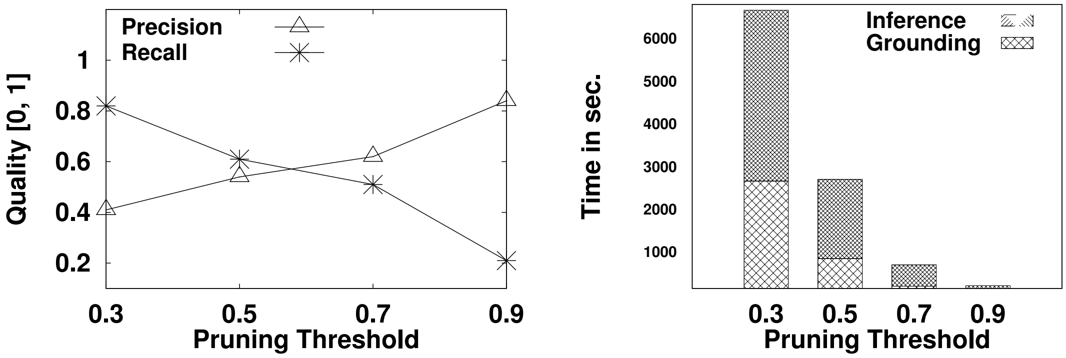
(b) No. of Rules vs. Exec. Time

Figure 3.10: Comparison of *Sya* with DeepDive using Step Function Rules.

time in both datasets. The main reason behind this performance gain is applying the concliques-based partitioning in the spatial Gibbs sampling algorithm (Section 3.5), 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.

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 3.7 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 3.10(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 3.10(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



(a) Threshold vs. Quality

(b) Threshold vs. Exec. Time

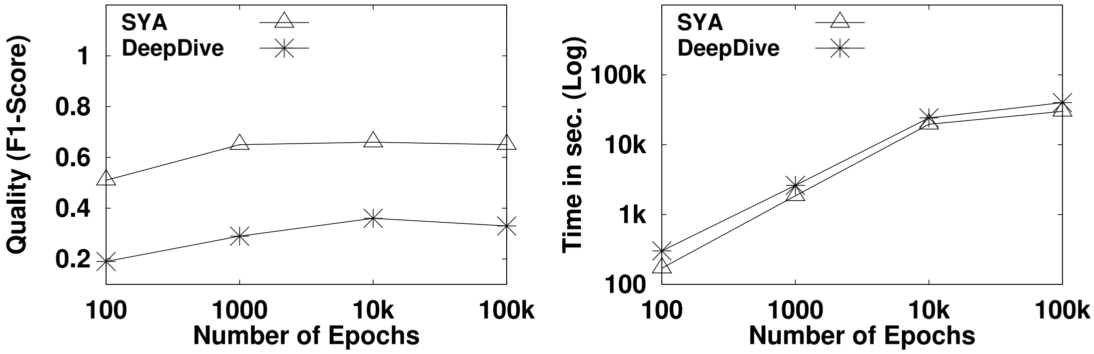
Figure 3.11: Effect of Pruning Threshold on Quality and Execution Time.

in the grounding phase to obtain 20% less F1-score compared to *Sya*, which is the best score achieved by DeepDive in our experiments.

Effect of Pruning Threshold

Figure 3.11(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 3.11(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 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 3.11(a).



(a) Epochs vs. F1-Score

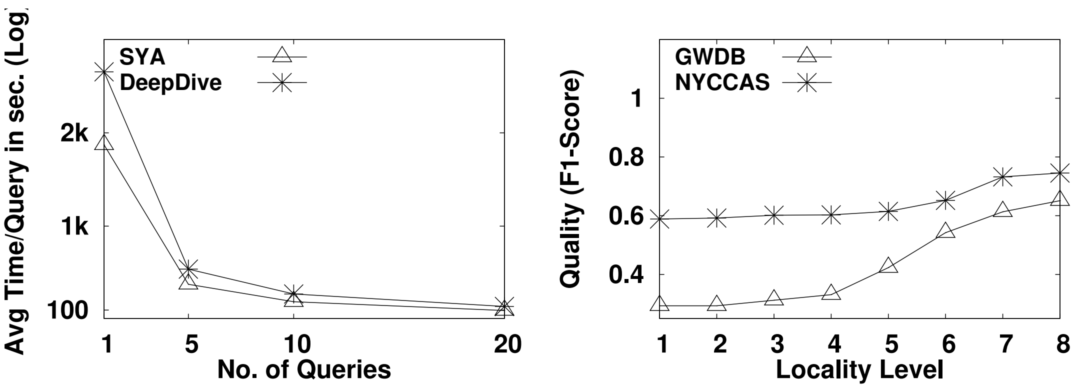
(b) Epochs vs. Inference Time

Figure 3.12: Effect of Inference Epochs on F1-Score and Inference Time.

Effect of Number of Inference Epochs

Figure 3.12(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 3.12(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 3.12(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 3.9(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.



(a) Incremental Inference

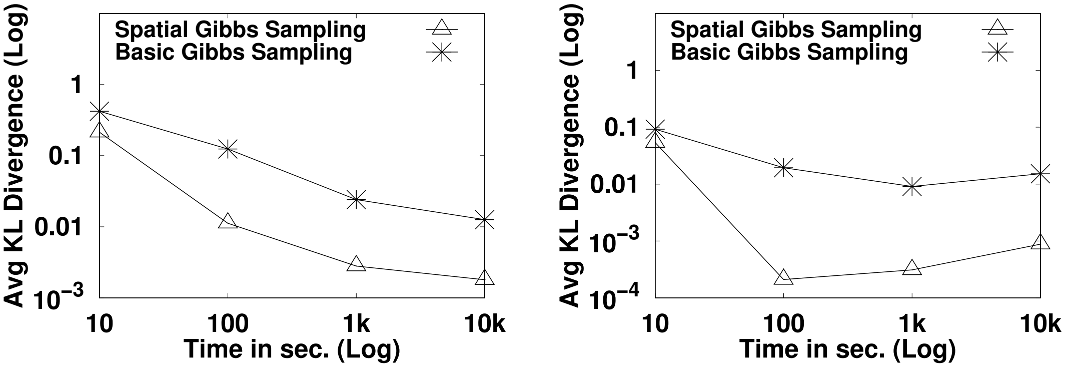
(b) Locality Level vs. Quality

Figure 3.13: Effect of Incremental Inference and Locality Level.

Effect of Incremental Inference and Locality Level

Figure 3.13(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 3.13(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 for the output knowledge bases.



(a) GWDB Dataset

(b) NYCCAS Dataset

Figure 3.14: Quality of Spatial Gibbs Sampling with Different Datasets.

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 [8, 99], 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 [103] between the estimated marginal probabilities using this algorithm and the true marginal probabilities provided by the ground truth. Figures 3.14(a) and 3.14(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 3.12(a).

3.7 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 [79, 80, 104] and crowdsourcing rules [76, 78, 105]), and *machine learning-based systems* (e.g., classification [106–114], maximum-a-posteriori models [84, 115, 116], probabilistic graphical models [117–119], Markov Logic Networks

(MLN) [2, 9, 36, 37, 43, 120], and deep learning [121]). We refer to these as "traditional" systems. The closest of these systems considering spatial attributes are [77], [122] and [123], 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. Recent research has focused on extracting special types of facts, such as subjective [124] and exceptional [125] facts. However, this line of research is out of the scope of our work.

- **Geo-Knowledge Bases.** Recent knowledge base systems have been proposed to extract facts about spatial entities (e.g., lakes) from Volunteered Geographic Information (VGI) [126] along with Semantic Geospatial Web [127] (see [128] for a comprehensive survey). For example, LinkedGeoData [129] assocaiates the spatial entities in the widely-used OpenStreetMap [130] dataset with their corresponding facts that exist in DBpedia [76]. In addition, a recent work has been focusing on the problem of entity alignment between knowledge bases with a special focus on spatial entities [131]. However, extracting and maintaining facts about spatial entities is a vastly different problem than we study in this chapter. In *Sya*, we extract a knowledge base of generic facts, yet, we exploit the spatial information, if any, to improve the output quality.

- **Knowledge Base Construction Languages.** A knowledge base construction system needs a high-level language that allows users to provide schema declaration and constuction logic rules. Existing languages are based on either object-oriented (e.g., UIMA [132], GATE [133]), reasoning (e.g., [134]), or datalog languages (e.g., DDlog [2], XLog [72], Lixto [135]). Datalog languages are considered the most popular amongst others in the literature because: (1) Datalog supports expressing horn clauses, which are the building blocks of inference rules, in a declarative way, and (2) Datalog syntax can be easily translated into SQL-like one, which makes it easy to be integrated with DBMS. Unfortunately, these languages do not support expressing spatial data types and predicates.

- **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 [8, 9], Markov chain Monte Carlo (MCMC) [7, 37, 66, 98, 136], belief propagation [137], lifted inference [138], or specialized algorithms of Markov Logic Network [4,139]. *Sya* provides a new variant of Gibbs sampling that adapts Concliques-based partitioning [88]. In our work, we show that the proposed variant is able to achieve high quality in the knowledge base construction process.

3.8 Conclusions

In this chaper, 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.

Chapter 4

TurboReg: A Framework for Scaling Up Spatial Logistic Regression Models

4.1 Introduction

Predicting the presence or absence of spatial phenomena in a certain geographical area is a crucial task in many scientific domains (e.g., Earth observations [140, 141], Epidemiology [142–144], Ecology [145, 146], Agriculture [147, 148] and Management [149]). For example, ornithologists would need to predict the presence or absence of a certain bird species across a certain area [150]. Meteorologists would need to predict the hurricane or tornado boundaries. Epidemiologist would need to understand the spread of diseases across various areas in the world. Typically, this is done by dividing the geographical space (e.g., the whole world) by a two-dimensional grid, where each grid cell is represented with a binary variable (i.e., takes either 0 or 1) indicating the presence or absence of the spatial phenomena in that grid cell, and a set of predictor variables (i.e., features) that help predicting the value of this binary variable. Then, the prediction problem at any grid cell is formulated as: *Given a set of predictors defined over this cell along with a set of observed or predicted values at neighbouring cells, predict the value of the binary variable at this cell.*

A common approach to solve the prediction problem is to build a standard logistic regression model [151, 152] that uses a logistic function to predict the value of each grid cell based on the values of predictors in the same grid cell. However, standard logistic regression models are deemed inappropriate for predicting spatial phenomena as they assume that neighboring locations are completely independent of each other. This is definitely not the case for spatial phenomena as neighboring locations tend to systematically affect each other [95].

As a result, spatial variants of logistic regression models (a.k.a., autologistic regression) were proposed to take into account the spatial dependence between neighboring grid cells [10, 11,49]. However, existing methods for autologistic regression (e.g., see [10,48,52,53]) are prohibitively computationally expensive for large grid data, e.g., fine-grained satellite images [140, 153], and large spatial epidemiology datasets [154]. For example, it could take about week to infer the autologistic model parameters using training data of only few gigabytes [10]. As a means of dealing with such scalability issues, existing techniques tend to sacrifice their accuracy through two simplified strategies: (1) Use only a small sample of the available training data, and (2) Only allow individual pairwise dependency between neighboring cells. For example, if a prediction cell variable C_1 depends on two neighboring cells C_2 and C_3, then current methods assume that cell C_1 depends on each of them individually, and hence define two pairwise dependency relations (C_1, C_2) and (C_1, C_3). Both approaches lead to significant inaccuracy and invalidate the use of autologistic regression for predicting spatial phenomena of current applications with large-scale training data sets.

In this chapter, we introduce *TurboReg*; a scalable framework for using autologistic models in predicting large-scale spatial phenomena. *TurboReg* does not need to sample training data sets. It can support prediction over grids of 85000 cells in 10 seconds. Moreover, *TurboReg* allows its users to define high degrees of dependency relations among neighbors, which opens the opportunity for capturing more precise spatial dependencies in regression. For example, for the case where a prediction cell variable C_1 depends on two neighboring cells C_2 and C_3, *TurboReg* is scalable enough to be able to define a ternary dependency relation (C_1, C_2, C_3), which gives much higher accuracy than having two independent binary relations.

TurboReg exploits Markov Logic Networks (MLN) [35] (a scalable statistical inference and learning framework) to learn the autologistic regression parameters in an accurate and efficient manner. Then, *TurboReg* aims to provide an equivalent first-order logic [14] representation to dependency relations among neighbors in autologistic models. This is necessary to accurately express the autologistic models using MLN. Since we focus on binary prediction variables, *TurboReg* transforms each neighboring dependency relation into a predicate with bitwise-AND operation on all variables involved in this relation. For example, a ternary dependency relation between neighboring variables C_1, C_2 and C_3 is transformed to $C_1 \wedge C_2 \wedge C_3$. This simple logical transformation allows non-expert users to express the dependency relations within autologistic models in a simple way without needing to specify complex models in a tedious detail.

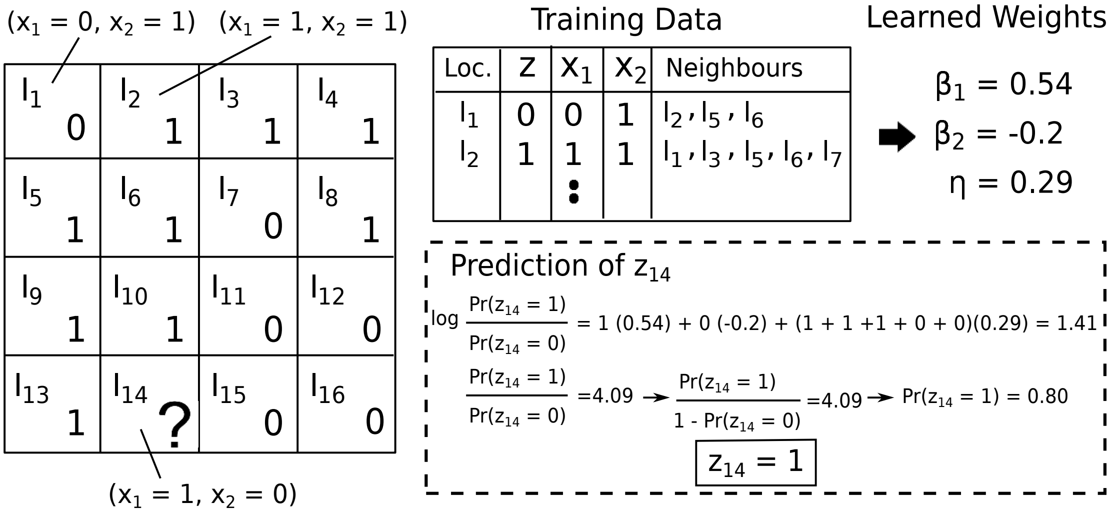
TurboReg proposes an efficient framework that learns the autologistic model parameters over MLN in a distributed manner. It employs a spatially-indexed learning graph structure, namely factor graph [7], along with an efficient weights optimization technique based on gradient descent optimization [75]. *TurboReg* represents the MLN bitwise-AND predicates using the spatially-indexed factor graph. Then, *TurboReg* runs multiple instances of learning algorithms in parallel, where each instance handles the learning process over exactly one factor graph partition. At the end, the obtained results from all learning instances are merged together to provide the final autologistic model parameters. Using the proposed framework, *TurboReg* converges to the optimal model parameters faster than the existing computational methods by at least three orders of magnitude while preserving the same parameters accuracy.

We experimentally evaluate *TurboReg* using a real dataset of the daily distribution of bird species [150], and a synthetic dataset about the crime types in Minneapolis, MN area. For each dataset, we compare the accuracy and scalability of the built autologistic models using *TurboReg* and a state-of-the-art open-source autologistic model computational method, namely ngspatial [51]. Our experimental evaluation shows that *TurboReg* is scalable to large-scale autologistic models compared to the existing techniques, while preserving high-level of accuracy in estimating the model parameters.

The rest of this chapter is organized as follows: Section 4.2 gives a brief background of autologistic regression models. Section 4.3 describes how autologistic regression is modeled using MLN. Section 4.4 gives an overview of *TurboReg*. Section 4.5 describes how the first-order logic predicates are generated for autologistic models. Section 4.6 provides details about the spatially-indexed factor graph structure. Section 4.7 illustrates the details of the weights learning phase. Section 4.8 provides an experimental analysis of *TurboReg*. Section 4.9 covers the related work to *TurboReg*, while Section 4.10 concludes the chapter.

4.2 Preliminaries of Autologistic Regression

This section provides a brief background along with a running example of the autologistic regression models. It also discusses our assumptions in *TurboReg*.



l_1 0	l_2 1	l_3 1	l_4 1
l_5 1	l_6 1	l_7 0	l_8 1
l_9 1	l_{10} 1	l_{11} 0	l_{12} 0
l_{13} 1	l_{14} ?	l_{15} 0	l_{16} 0

Loc.	Z	X$_1$	X$_2$	Neighbours
l_1	0	0	1	l_2, l_5, l_6
l_2	1	1	1	l_1, l_3, l_5, l_6, l_7
		⋮		

$$\beta_1 = 0.54 \quad \beta_2 = -0.2 \quad \eta = 0.29$$

$$\log \frac{Pr(z_{14} = 1)}{Pr(z_{14} = 0)} = 1\ (0.54) + 0\ (-0.2) + (1 + 1 + 1 + 0 + 0)(0.29) = 1.41$$

$$\frac{Pr(z_{14} = 1)}{Pr(z_{14} = 0)} = 4.09 \rightarrow \frac{Pr(z_{14} = 1)}{1 - Pr(z_{14} = 0)} = 4.09 \rightarrow Pr(z_{14} = 1) = 0.80$$

$$\boxed{z_{14} = 1}$$

Figure 4.1: An Example on Autologistic Regression.

Autologistic regression builds a regression model that predicts the value of a binary random variable (i.e., prediction variable that takes either 0 or 1) at a certain location based on a set of predictors (i.e., features that help in the prediction process) at the same location and a set of observed predictions from variables at neighbouring locations (i.e., spatial dependence). Formally, autologistic regression models assume a set of n binary prediction variables $\mathcal{Z} = \{z_1, ..., z_n\}$ (i.e., $z_i \in \{0, 1\}$) at n locations $\mathcal{L} = \{l_1, ..., l_n\}$, and a set of m predictor variables $\mathcal{X}(i) = \{x_1(i), ..., x_m(i)\}$ where the value of each predictor variable $x_j(i)$ is a function of location l_i (e.g., a predicator about the existence of water which could have a different value for each location), and each location l_i has a set of neighbouring locations $\mathcal{N}_i$. Given a specific location l_i, the conditional probability of prediction variable z_i given the values of current predictor variables $\mathcal{X}$ and the neighbouring prediction variables $\mathcal{Z}_{\mathcal{N}_i}$ can be estimated as follows [10, 11]:

$$\log \frac{Pr(z_i = 1 \mid \mathcal{X}(i), \mathcal{Z}_{\mathcal{N}_i})}{Pr(z_i = 0 \mid \mathcal{X}(i), \mathcal{Z}_{\mathcal{N}_i})} = \sum_{j=1}^{m} \beta_j x_j(i) + \eta \sum_{k \in \mathcal{N}_i} z_k \qquad (4.1)$$

where the weights $\beta = \{\beta_1, ..., \beta_m\}$ and η form the model parameters $\theta = \{\beta, \eta\}$. The objective of our work is to learn the values of θ from previous observations (i.e., training data) of predictions and predictors at locations $\mathcal{L}$, in a scalable and efficient manner. Note that the

Predicate	Weight
f_1: z_1 ^ x_1	β_1
f_2: z_1 ^ z_2	η
f_3: z_1 ^ z_3	η
f_4: z_2 ^ x_1	β_1
f_5: z_2 ^ z_4	η
f_6: z_3 ^ x_1	β_1
f_7: z_3 ^ z_4	η
f_8: z_4 ^ x_1	β_1

Weighted Logical Predicates

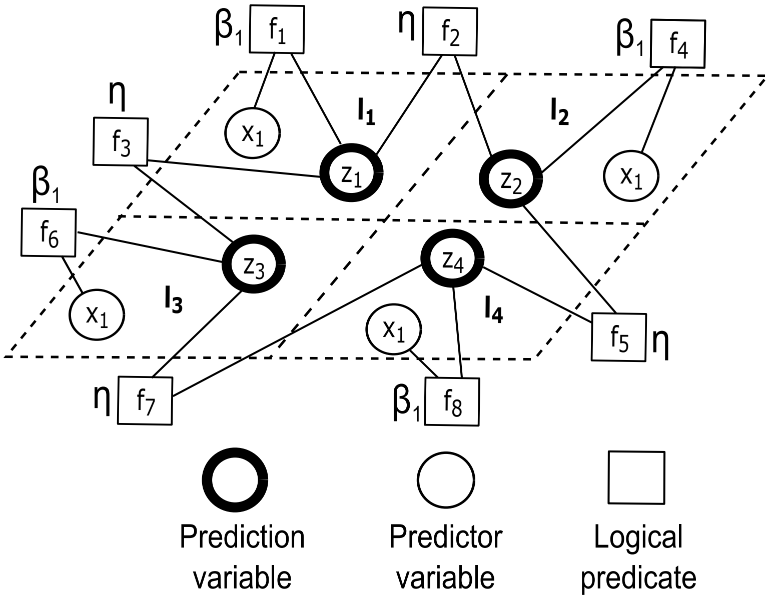


Figure 4.2: An Equivalent MLN Representation to An Autologistic Regression Model (logical predicates and their factor graph).

values of θ are shared among the whole locations $\mathcal{L}$.

Assumptions. In general, predictor variables $\mathcal{X}$ can be either binary, categorical, or continuous. However, we focus only on binary predictors (i.e., $x_m(i) \in \{0, 1\}$). The extension to categorical and continuous cases is intuitive as well, but, out of scope of *TurboReg*.

Example. Figure 4.1 shows a numerical example of autologistic regression. In this example, we have a 4 x 4 grid (i.e., 16 cells), where each cell l_i has a prediction variable z_i and two predictor variables $x_1(i)$ and $x_2(i)$. The model parameters β_1 and β_2 are trained by observations from all locations except l_{14} which is unknown (i.e., needs to be predicted). The example also shows the calculations to predict the value of z_{14} using the learned parameters.

4.3 Autologistic Regression via Markov Logic Networks

In this section, we describe how the MLN framework is exploited to efficiently solve the autologistic regression problem. We start by discussing an MLN-based model for the basic autologistic regression (Section 4.3.1), and providing its theoretical foundation (Section 4.3.2). Then, we extend this MLN-based model in case of more complicated scenarios (Section 4.3.3).

4.3.1 MLN-based Autologistic Models

To represent an autologistic model using MLN, the model should have the two main properties mentioned in Section 2.3. Obviously, the first property is satisfied because all prediction and predictor variables (i.e., $\mathcal{Z}$ and $\mathcal{X}$) are already binary. However, to achieve the second property, the model is required to have a set of equivalent constraints (i.e., logical predicates) that capture the autologistic regression semantics. As shown in Equation 4.1, there are two types of regression terms: (1) *predictor-based* terms $\{\beta_j x_j(i) \mid j = 1, .., m\}$ where m is the number of predictors at any location l_i, and (2) *neighbour-based* terms $\{\eta z_k \mid k \in \mathcal{N}_i\}$ where $\mathcal{N}_i$ is the set of neighbours at location l_i. *TurboReg* provides an equivalent weighted first-order logic predicate to each regression term, either predictor-based or neighbour-based, that preserves the semantic of autologistic regression and can be represented with MLN as well. For each prediction variable z_i at location l_i, each *predictor-based* regression term $\beta_j x_j(i)$ has an equivalent bitwise-AND predicate defined over z_i and $x_j(i)$ (i.e., $z_i \wedge x_j(i)$) with weight β_j. Similarly, each *neighbour-based* regression term ηz_k has an equivalent bitwise-AND predicate defined over z_i and z_k (i.e., $z_i \wedge z_k$) with weight η. The theoretical foundation of the proposed MLN-based autologistic model is described in the Section 4.3.2.

Note that, using the proposed model, the autologistic regression parameters $\theta = \{\beta, \eta\}$ are translated into a set of weights $\mathcal{W}$ of MLN constraints (i.e., proposed equivalent bitwise-AND predicates), and hence learning the autologistic model parameters θ becomes equivalent to learning the values of $\mathcal{W}$ in MLN (See Section 2.2).

Example. Figure 4.2 shows an example of translating an autologistic regression model with one predictor variable x_1 (i.e., log $\frac{Pr(z_i=1|\mathcal{X},\mathcal{Z}_{\mathcal{N}_i})}{Pr(z_i=0|\mathcal{X},\mathcal{Z}_{\mathcal{N}_i})} = \beta_1 x_1(i) + \eta \sum_{k \in \mathcal{N}_i} z_k$) into an equivalent MLN. The model is built for a 4-cells grid, where the neighbourhood $\mathcal{N}_i$ of any cell l_i is assumed to be cells that share edges with l_i (i.e., first-order neighbourhood). The model is first translated into a set of 8 bitwise-AND predicates with two weights β_1 and η. Then, these predicates are translated into a factor graph which can be used to learn the weights β_1 and η. Note that duplicate predicates that come from neighbouring variables are removed to avoid redundancy (e.g., the neighbouring variables z_1 and z_2 have two equivalent $z_1 \wedge z_2$ and $z_2 \wedge z_1$ neighbour-based predicates, respectively, however, we keep only one of them).

4.3.2 Theoretical Foundation of TurboReg using MLN

Theorem 1 *Given an autologistic model with a set of n prediction variables $\mathcal{Z} = \{z_1, ..., z_n\}$ defined over n locations $\mathcal{L} = \{l_1, ..., l_n\}$, a set of m weighted predictor variables $\beta\mathcal{X}(i) = \{\beta_1 x_1(i), ..., \beta_m x_m(i)\}$ at each location i and neighbouring weight η, there is an equivalent Markov Logic Network (MLN) to this model, if and only if: (1) each predictor-based regression term $\beta_j x_j(i)$ at location l_i has an equivalent bitwise-AND predicate $z_i \wedge x_j(i)$ with weight β_j. (2) each neighbour-based regression term ηz_k at location l_i has an equivalent bitwise-AND predicate $z_i \wedge z_k$ with weight η.*

Proof. Assume a model that consists of $n + nm$ binary random variables $\mathcal{V} = \{\mathcal{Z}, \mathcal{X}\} = \{z_1, ..., z_n, x_1(1), ..., x_m(n)\}$. In addition, assume a set of constraints $\mathcal{F} = \{F_1, ..., F_n\}$ are defined over variables $\mathcal{V}$, where constraints F_i at location l_i consist of two subsets of constraints:

- a set of m bitwise-AND predicates $\{z_i \wedge x_j(i) \mid j = 1, ..., m\}$ with β weights (each predicate corresponds to a predictor-based regression term).
- a set of s_i bitwise-AND predicates $\{z_i \wedge z_k \mid k \in \mathcal{N}_i\}$ with η weight (each predicate corresponds to a neighbour-based regression term) where s_i is the size of neighbouring locations $\mathcal{N}_i$ of location l_i.

Based on these assumptions, the model satisfies the two main properties in Section 2.3 that are needed to represent it using MLN, and hence its joint probability distribution over $\mathcal{V}$ is estimated using Equation 2.7.

Since $\mathcal{Z}$ and $\mathcal{X}$ are binary random variables, the evaluation of any bitwise-AND predicate over them can be represented as a mathematical multiplication (i.e., the value of $z_i \wedge x_j(i)$ is $z_i x_j(i)$ and the value of $z_i \wedge z_k$ is $z_i z_k$). As a result, the joint probability distribution of $\mathcal{V}$ from Equation 2.7 becomes as follows:

$$Pr(\mathcal{V} = v) = Pr(\mathcal{Z}, \mathcal{X}) = \frac{1}{C} \exp\Big(\sum_{i=1}^{n}\sum_{j=1}^{m} \beta_j z_i x_j(i) + \eta \sum_{i=1}^{n}\sum_{k\in\mathcal{N}_i} z_i z_k\Big) \tag{4.2}$$

Based on Equation 4.2, the conditional probability distribution of any prediction variable z_i at location l_i given the predictor variables $\mathcal{X}(i)$ at l_i and its neighbouring prediction variables $\mathcal{Z}_{\mathcal{N}_i}$ can be estimated as:

$$Pr(z_i = 1 \mid \mathcal{X}(i), \mathcal{Z}_{\mathcal{N}_i}) = \frac{1}{C} \exp\Big(\sum_{j=1}^{m} \beta_j z_i x_j(i) + \eta \sum_{k \in \mathcal{N}_i} z_i z_k\Big) \tag{4.3}$$

By substituting with possible values of z_i (i.e., either 1 or 0) in Equation 4.3, we can obtain the ratio between the conditional probabilities of $z_i = 1$ and $z_i = 0$ as follows:

$$\begin{aligned} \frac{Pr(z_i = 1 \mid \mathcal{X}(i), \mathcal{Z}_{\mathcal{N}_i})}{Pr(z_i = 0 \mid \mathcal{X}(i), \mathcal{Z}_{\mathcal{N}_i})} &= \frac{\exp\Big(1 \times \sum_{j=1}^{m} \beta_j x_j(i) + 1 \times \eta \sum_{k \in \mathcal{N}_i} z_k\Big)}{\exp\Big(0 \times \sum_{j=1}^{m} \beta_j x_j(i) + 0 \times \eta \sum_{k \in \mathcal{N}_i} z_k\Big)} \\ &= \frac{\exp\Big(\sum_{j=1}^{m} \beta_j x_j(i) + \eta \sum_{k \in \mathcal{N}_i} z_k\Big)}{\exp\Big(0\Big)} \\ &= \exp\Big(\sum_{j=1}^{m} \beta_j x_j(i) + \eta \sum_{k \in \mathcal{N}_i} z_k\Big) \end{aligned} \tag{4.4}$$

By taking the $\log$ value of both LHS and RHS of Equation 4.4, we obtain the autologistic model defined in Equation 4.1. This means that the assumed model at the beginning, which can be represented with MLN, is equivalent to the basic autologistic model.

4.3.3 Generalized Autologistic Models

Some applications assume models with more generalized neighbour-based regression terms $\{\eta G(z_{k_1}, ..., z_{k_d}) \mid k_1, ..., k_d \in \mathcal{N}_i\}$ (i.e., complex spatial dependence), where the regression term has a function defined over neighbouring prediction variables $G(z_{k_1}, ..., z_{k_d})$, and not just their sum as in Equation 4.1 (e.g., Ecology [155] and Mineral Exploration [156]).

Existing methods can not compute autologistic models with generalized regression terms because of their prohibitively expensive computations, such as high-order matrix multiplications [10]. In contrast, the MLN-based autologistic model can be easily extended to find an equivalent combination of first-order logic predicates [14] for any generalized regression term, as long as the function $G(z_{k_1}, ..., z_{k_d})$ holds logical semantics. For example, if prediction variable z_1 at location l_1 has a generalized regression function $G(z_2, z_3)$ over neighbours z_2 and z_3 which constraints the value of z_1 to be 1 only if both values of z_2 and z_3 are 1 at the same time,

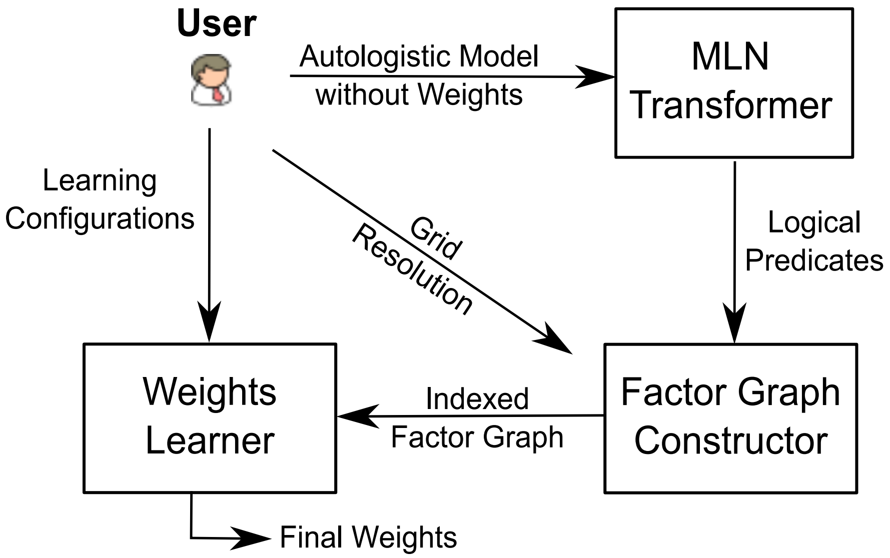


Figure 4.3: *TurboReg* System Architecture.

then *TurboReg* would translate this into an equivalent bitwise-AND predicate $z_1 \wedge z_2 \wedge z_3$. As another example, if $G(z_2, z_3)$ constraints the value of z_1 to be 1 only if the either z_2 or z_3 is 1, then it can be translated into a predicate $z_1 \wedge (z_2 \vee z_3)$ that has a combination of bitwise-AND and bitwise-OR. Our experiments show that handling generalized regression terms using the MLN-based model increases the learning accuracy while not affecting the scalability performance (See Section 4.8).

4.4 Overview of TurboReg

Figure 4.3 depicts the system architecture of *TurboReg*. It includes three main modules, namely, *MLN Transformer*, *Factor Graph Constructor*, and *Weights Learner*, described as follows:

MLN Transformer. This module receives the autologistic regression model from any *TurboReg* user and generates a set of bitwise-AND predicates of an equivalent MLN. It employs an efficient logic programming framework, called DDlog [2], to produce predicates in a scalable manner. Details are in Section 4.5.

Factor Graph Constructor. This module prepares the input for the *Weights Learner* module by building a spatially-indexed factor graph out of the generated bitwise-AND predicates. The factor graph is partitioned using a flat grid index, where each grid cell has a graph index for its factor graph part. Details are in Section 4.6.

Weights Learner. This is the main module in *TurboReg* which efficiently learns the weights that are encoded in the spatially-indexed factor graphs. These weights represent the autologistic model parameters. It takes the built factor graph along with learning configurations (e.g., number of learning epochs) as input, and produces the final values of weights $\theta = \{\beta, \eta\}$. In this module, *TurboReg* provides a scalable variation of gradient descent [75] technique, that is highly optimized for learning the autologistic model parameters. Details are in Section 4.7.

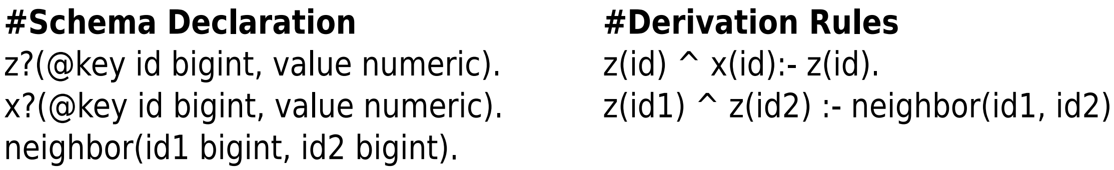
4.5 MLN Transformer

The first step in *TurboReg* is to generate a set of equivalent logical predicates for the different regression terms in the autologistic model. However, this step is challenging if : (1) the model has a large number of prediction variables $\mathcal{Z}$ (i.e., large 2-dimensional grid) and/or predictor variables $\mathcal{X}$ (e.g., large number of synthetic features), which results in generating a large number of neighbour-based and predictor-based predicates at the end. (2) the model has very complicated generalized regression terms, which are translated into predicates with large number of combinations of first-order logic symbols (e.g., bitwise-AND, bitwise-OR, and imply).

To remedy this challenge, *TurboReg* uses DDlog [2], an DBMS-based logic programming framework, to generate equivalent predicates for any autologistic model in a scalable manner. DDlog takes advantage of the scalability provided by DBMS when generating large number or combinations of predicates. It provides users with a high-level declarative language to express logical predicates using few template rules. These rules are then translated into SQL queries and applied against database relations of variables (e.g., $\mathcal{Z}$ and $\mathcal{X}$) to instantiate the actual set of predicates. DDlog has been widely adopted in many applications due to its usability and efficiency (e.g., knowledge bases [2] and data cleaning [3]).

Example. Figure 4.4 shows an example of using DDlog to express the bitwise-AND predicates of the autologistic model in Figure 4.2. DDlog has two types of syntax; *schema declaration* and *derivation rules*. Schema declaration defines the relational schema of variables that appear in predicates. For example, prediction variables $\mathcal{Z}$ and predictor variables $\mathcal{X}$ are stored in relations $z?$ and $x?$, respectively, where any row in each relation corresponds to one variable and stores its location ID and its to-be-predicted value in attributes id and $value$, respectively.

Note that variable relations are differentiated from normal relations with a question mark at the end of their names. Derivation rules are considered templates to instantiate predicates. In



```
#Schema Declaration
z?(@key id bigint, value numeric).
x?(@key id bigint, value numeric).
neighbor(id1 bigint, id2 bigint).
```

```
#Derivation Rules
z(id) ^ x(id):- z(id).
z(id1) ^ z(id2) :- neighbor(id1, id2)
```

Figure 4.4: Example of Using DDlog to Generate Bitwise-AND Predicates for An Autologistic Regression Model.

this example, the first derivation rule is a template for bitwise-AND predicates coming from the predictor-based regression terms (i.e., f_1, f_4, f_6 and f_8 in Figure 4.2), where the body of rule (i.e., right side after symbol ":-") specifies that a predicate is defined over any z and x only if they have same location id (i.e., selection criteria). During execution, this rule is translated into a hash join between relations z and x with selection predicate over id. Similarly, the second derivation rule is a template for predicates corresponding to the neighbour-based regression terms (i.e., f_2, f_3, f_5 and f_7 in Figure 4.2), where a predicate is defined for each individual pair of neighbouring predication variables.

4.6 Factor Graph Constructor

Figure 4.5 depicts the organization of a spatially-indexed factor graph for the predicates that are generated in Figure 4.2. The index is composed of two main layers, namely, *neighbourhood* layer and *graph* layer, described as follows:

4.6.1 Neighbourhood Index Layer

The neighbourhood index layer is basically a two-dimensional index on the given factor graph. There is already a rich literature on two-dimensional index structures, classified into two categories: *Data-partitioning* index structures (e.g., R-tree [96]) that partition the data over the index and *space-partitioning* index structures (e.g., Quadtrees [157]) that partition the space.

In *TurboReg*, we decided to go with the Grid Index [158] as an example of the space-partitioning data structures because it aligns with the nature of spatial phenomena that are predicted over grids. Having said this, *TurboReg* can accommodate other two-dimensional index structures as a replacement of our grid index. Each grid cell in the neighbourhood layer keeps a graph index for its factor graph part. Figure 4.5 gives an example of a neighbourhood index layer as a 2-cells grid (i.e., C_1 and C_2), where C_1 contains the factor graph part corresponding to predicates in locations l_1 and l_3, and C_2 holds predicates in locations l_2 and l_4. *TurboReg* takes the grid resolution as input from the user.

4.6.2 Graph Index Layer

Each cell in the two-dimensional neighbourhood grid points to two indexes of *variables* and *predicates*. Together, these two indexes form the factor graph part in this cell.

Variables Index. This index contains all predication and predictor variables that exist in the grid cell. Each node in the index corresponds to one variable, and points to a list that has three types of information (1) location as a first element in list, (2) value (i.e., 1 or 0) as a second element in the list and (3) predicates that this variable appear in, which are stored as a set of pairs in the rest of list. Each pair consists of a pointer to a predicate in the *predicates* index, and its associated weight. Figure 4.5 shows the details of variable z_1 in the variables index.

Predicates Index. This index contains all predicates that defined over variables in this grid cell. Each node in the index corresponds to one predicate and has a list of pointers to variables that appear in this predicate. Figure 4.5 shows the details of predicate f_6 in the predicates index. Note that we replicate predicates that have variables in two different grid cells (e.g., f_7 is duplicated in C_1 and C_2 because it has variable $z_3 \in C_1$ and $z_4 \in C_2$).

4.7 Weights Learner

This is the most important module in *TurboReg*, which takes the spatially-indexed factor graph along with learning configurations from user and returns the final weights $\theta = \{\beta, \eta\}$ of the autologistic model. The main idea is to incrementally converge to weights that maximize the satisfaction of bitwise-AND predicates represented in the factor graph. For example, if the predicate $z_i \wedge x_i$ is satisfied, then the current value of its weight η should be rewarded (i.e., should be increased), otherwise should be punished (i.e., should be decreased). To that end, we adapt a technique that punishs and rewards weights using gradient descent optimization [75]. To test satisfaction of any bitwise-AND predicate in autologistic regression, we suggest to substitute in Equation 4.1 with the values of variables that appear in the predicate along with its weight. The details of algorithms that implement this idea are described below.

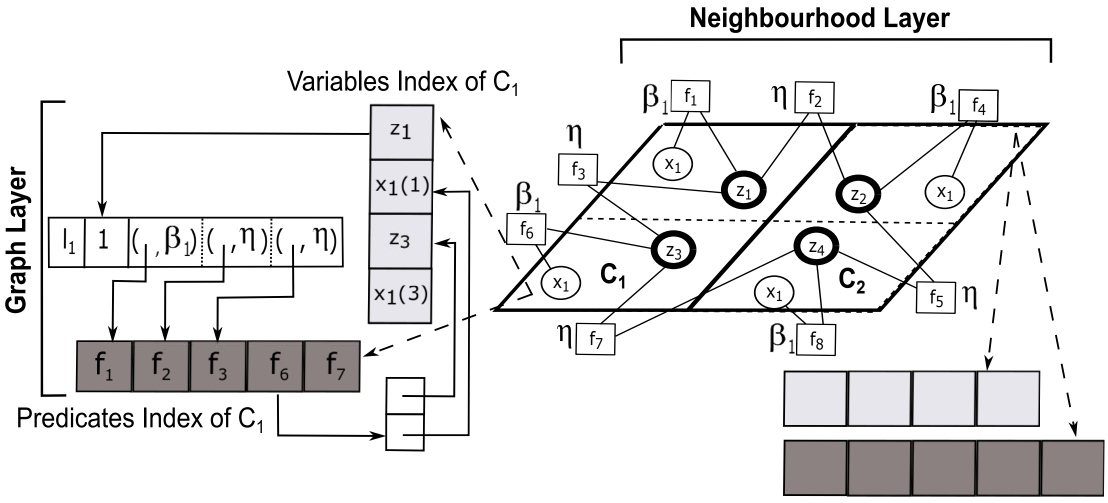


Figure 4.5: Example of Spatially-indexed Factor Graph.

LEARNWEIGHTS Algorithm. Algorithm 3 depicts the pseudo code for our scalable weights learner that takes the following four inputs: the spatially-partitioned factor graph C, the number of learning instances S that can run in parallel, the number of learning iterations E needed to converge to the final values of weights, and the step size α which is a specific parameter for the optimization algorithm 4 that will be described later. The algorithm keeps track of the current best values of weights through variables: β and η, initialized by random values. The algorithm then starts by computing the number of learning epochs that can be handled per each learning instance and stores it in variable e. Note that e represents the actual number of learning epochs that run sequentially because different learning instances execute in parallel. Each of these learning instances then starts to process one learning epoch in parallel (i.e., S learning epochs are running simultaneously). In such learning epoch, we learn an optimal instance of weights β_s and η_s, where these values are incrementally learned from variables in factor graph using UPDATEWEIGHTS function (Line 9 in Algorithm 3)(details of this function are described in Algorithm 4). To reduce the learning latency, we process the variables from different factor graph partitions in parallel (Lines 5 to 10 in Algorithm 3). After all learning instances finish their current learning epoch, we set the values of β and η with the average of the obtained weights from these instances and then proceed to another learning epoch with the new weights. We repeat this process e times and then return the final values of weights.

Algorithm 3 Function LEARNWEIGHTS (FactorGraphCells C, LearningInstances S, LearningEpochs E, StepSize α)

```
$\beta \leftarrow$ Random, $\eta \leftarrow$ Random
$e \leftarrow \frac{E}{S}$ /* Num. of Learning Epochs Per Instance*/
while $e \neq 0$ do
    for all $s \in \{1, 2, ..., S\}$ do in parallel
        for all $c \in C$ do in parallel
            $\mathcal{V}_c \leftarrow$ Variables index in cell $c$
            $\mathcal{P}_c \leftarrow$ Predicates index in cell $c$
            for each $v_i \in \mathcal{V}_c$ do
                UPDATEWEIGHTS ($v_i$, $\mathcal{V}_c$, $\mathcal{P}_c$, $\alpha$) (Algorithm 4)
            end for
    $\beta \leftarrow \frac{\sum_{s=1}^{S} \beta_s}{S}$, $\eta \leftarrow \frac{\sum_{s=1}^{S} \eta_s}{S}$
    $e - -$
end while
return $\beta$ and $\eta$
```

UPDATEWEIGHTS Algorithm. Algorithm 4 gives the pseudo code for our weights optimizer that applies gradient descent optimization [75] technique to incrementally update the values of weights given a certain variable v_i (either prediction or predictor). The main idea is to punish or reward current weights based on their performance in correctly estimating the prediction value z_i at location l_i where variable v_i belongs to. The algorithm takes the following inputs; a variable v_i, the variables $\mathcal{V}$ and predicates $\mathcal{P}$ indexes in the grid cell containing v_i (i.e., graph index), and a step size α that controls the amount of punishing/rewarding during the optimization process. The algorithm keeps track of the current status of whether weights need to be punished or rewarded weights through variable g, where it takes either 1 in case of rewarding or -1 in case of punishing, and is initialized by 1. The algorithm starts by estimating the prediction $\hat{z_i}$ at location l_i that contains v_i using Equation 4.1. If the estimation $\hat{z_i}$ never matches the observed prediction value z_i from training data, then we set the status g to -1 (i.e., the associated weight with current variable v_i needs to be punished), otherwise the status remains rewarding. In case v_i is a predictor variable $x_j(i)$, we only update its associated weight β_j by evaluating the gradient descent equation using current values of g and α (Line 7 in Algorithm 4) and jump to the end of algorithm. In case v_i is the prediction variable z_i itself, we apply gradient descent optimization on all weights β associated with its predictors (Lines 9 to 11 in Algorithm 4), and on weight η associated with neighbouring predicates (Lines 14 to 24 in Algorithm 4) as well.

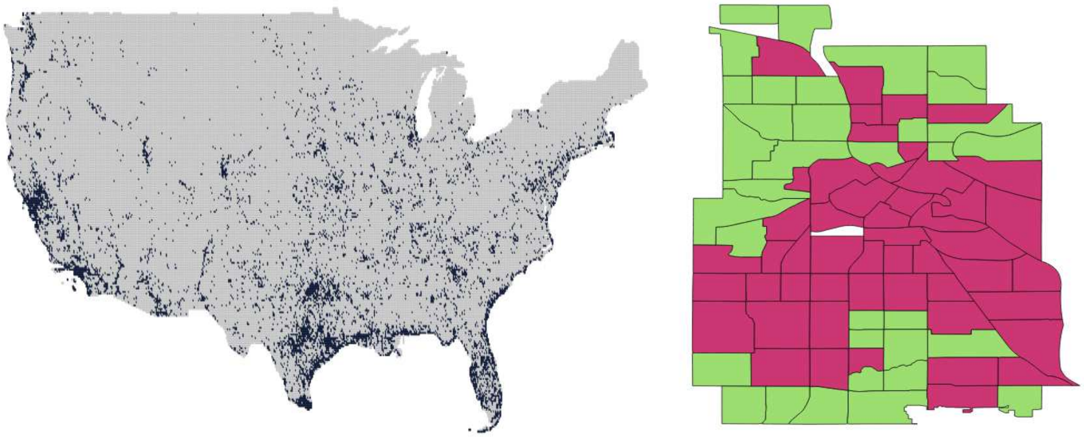
Algorithm 4 Function UPDATEWEIGHTS (Variable v_i, VariablesIndex $\mathcal{V}$, PredicatesIndex $\mathcal{P}$, StepSize α)

```
$l_i \leftarrow \mathcal{V}[v_i]$.location, $g \leftarrow 1$ /* Gradient Value */
$\hat{z_i} \leftarrow$ Prediction at $l_i$ using $\beta$ and $\eta$ (Equation 4.1)
if $\mathcal{V}[v_i]$.value $\neq \hat{z_i}$ then
    $g \leftarrow$ -1
end if
if $v_i$ is any predictor variable $x_j(i) \in \mathcal{X}(i)$ then
    $\beta_j \leftarrow \beta_j$ + $\alpha$ $g$ /* Gradient Descent on $\beta_j$*/
else
    for each $\beta_j \in \beta$ do
        $\beta_j \leftarrow \beta_j$ + $\alpha$ $g$ /* Gradient Descent on $\beta_j$*/
    end for
end if
if $v_i$ is prediction variable $z_i$ then
    for each $p \in \mathcal{P}[v_i]$ do
        if $p$ is a neighbour-based predicate then
            $\hat{z_k} \leftarrow$ Prediction at neighbour $l_k$ in $p$ using $\beta$ and $\eta$
            if $\mathcal{V}[v_k]$.value $\neq \hat{z_k}$ then
                $g \leftarrow$ -1
            else
                $g \leftarrow 1$
            end if
            $\eta \leftarrow \eta$ + $\alpha$ $g$ /* Gradient Descent on $\eta$ */
        end if
    end for
end if
```

Complexity. The complexity of the aforementioned algorithms can be estimated as $O(\frac{E}{S}\frac{(n^2+nm)}{C})$ where n is number of predictions, m is number of predictors, C is number of factor graph partitions, E is number of learning epochs and S is number of learning instances. This complexity can be further approximated to be $O(\frac{E}{S}\frac{(n^2)}{C})$. Note that we assume having SC working threads to process C factor graph partitions in each of the S learning instances in parallel.

4.8 Experiments

In this section, we experimentally evaluate the accuracy and scalability of *TurboReg* in building autologistic models (i.e., learning their weights). We compare the performance of *TurboReg*



(a) Ebird Data

(b) MNCrime Data

Figure 4.6: Datasets Used in *TurboReg* Experiments.

with ngspatial [51], a state-of-the-art open-source package for autologistic model implementations. Specifically, we compare our performance with the most accurate algorithm in ngspatial that employs bayesian inference using Markov Chain Monte Carlo (MCMC) [10]. We extensively investigate the accuracy and scalability of both systems under different parameters including grid sizes, learning epochs, and neighbourhood structures.

4.8.1 Experimental Setup

Datasets. All experiments are based on the following two grid datasets:

- *Ebird* dataset [150], which is a real dataset of the daily distribution of a certain bird species, namely Barn Swallow, over North America. Each grid cell holds a predication of the bird existence in the cell or not. Figure 4.6(a) shows the Ebird data distribution, where blue dots refer to cells with bird existence. We generate eight versions of this dataset with different grid sizes, ranging from 250 to 84000 cells, to be used during most of our experiments. This dataset has three binary predictors including whether number of bird observers high or not, whether the observing duration is long or not, and whether observers cover large spatial area or not.

- *MNCrimes* dataset, which is a synthetic dataset about predicting the existence of bike theft crime in 87 neighborhoods in Minneapolis. Figure 4.6(b) shows the MNCrimes data distribution, where red and green cells refer to the crime existence and non-existence, respectively. This grid is constructed based on three public datasets about Minneapolis neighbourhoods [159], census [160] and crime incidents [159]. It also uses the information about other 11 crime types as binary predictors.

In both Ebird and MNCrimes datasets, we randomly select 15% of the grid cells as testing data, and use the rest 85% for training. We use Ebird dataset in all experiments, except the experiment in the last Figure 4.10(b) which uses MNCrimes dataset.

Parameters. Unless otherwise mentioned, Table 4.1 shows the default settings of both Ebird and MNCrimes datasets. Note that we use small number of grid cells as a default value, because the ngspatial technique fails in large cases. However, we have standalone experiments to show the scalability of *TurboReg* with large number of grid cells. Table 4.2 also shows the default learning configurations that are used with *TurboReg* and ngspatial. In most of experiments, we run two variations of our system: the basic *TurboReg* that has pairwise neighbourhood relationships (i.e., neighbourhood degree of 1), and another generalized variation with 8-ary neighbourhood relationships (i.e., neighbourhood degree of 8), referred to as *G-TurboReg-8* (See Section 4.3.3). In *G-TurboReg-8*, each predication has a bitwise-AND predicate over the whole 8 neighbours surrounding it. In case of ngspatial, we set the default standard deviation α of any bayesian prior distributions with the recommended value 1000 as in their documentation [51].

Environment. 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.

Metrics. In all experiments, we use the total running time of learning weights as a scalability evaluation metric, and the ratio of correctly predicted cells to the total number of test cells as an accuracy evaluation metric.

4.8.2 Experimental Results

Effect of Grid Size

In this section, we compare the performance, both scalability and accuracy, of basic *TurboReg* and *G-TurboReg-8* with ngspatial, while having five different sizes of prediction grids.

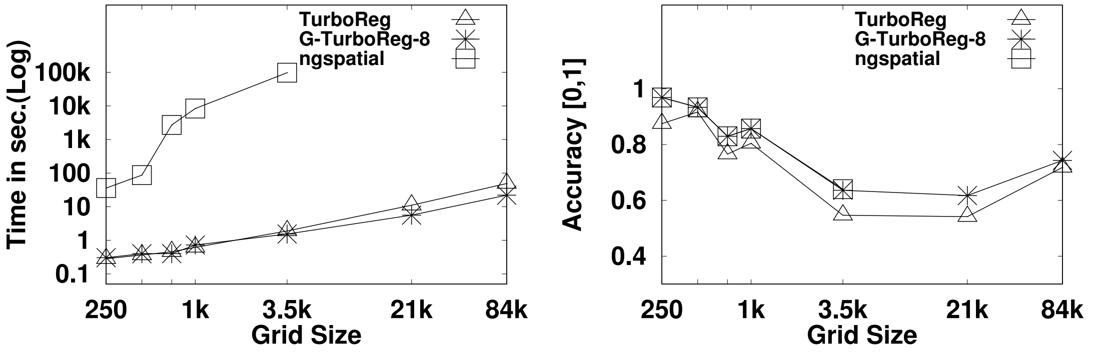
Figure 4.7(a) shows the running time for each algorithm while scaling the grid size from

Parameter	Default Value
Grid Training Size (Ebird)	860 Cells
Grid Testing Size (Ebird)	140 Cells
Number of Predictors (Ebird)	3
Grid Training Size (MNCrimes)	72 Cells
Grid Testing Size (MNCrimes)	12 Cells
Number of Predictors (MNCrimes)	11

Table 4.1: Dataset-specific Parameters.

Parameter	Default Value
Learning Epochs E	1000
Neighbourhood Degree D	1, 8
Step Size α (*TurboReg*)	0.001
Number of Threads (*TurboReg*)	7
Factor Graph Partitions (*TurboReg*)	200
Standard Deviation σ (ngspatial)	1000

Table 4.2: Learning-specific Parameters.

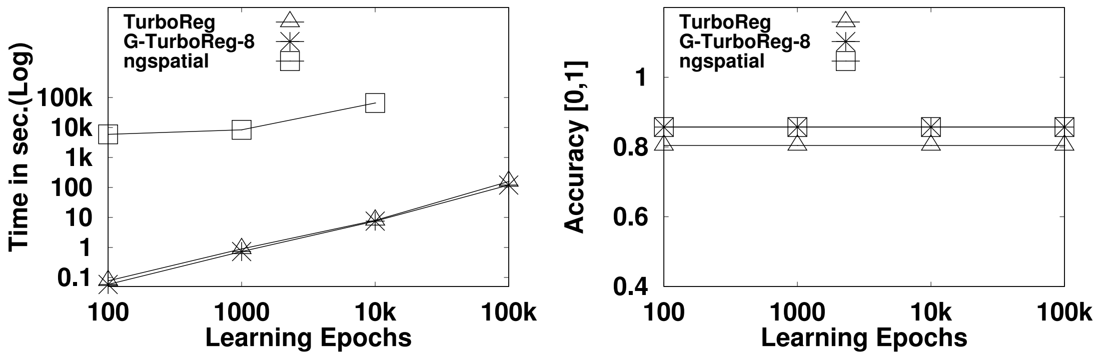


(a) Grid Size vs. Scalability

(b) Grid Size vs. Accuracy

Figure 4.7: Effect of Grid Size on Scalability and Accuracy.

250 to 84k cells. For the five grid sizes, both *TurboReg* and *G-TurboReg-8* were able to significantly reduce the running time compared to ngspatial. Specifically, both *TurboReg* variants and ngspatial have an average running time of 6 seconds and 6 hours, respectively. This means that *TurboReg* has at least three orders of magnitude reduction in the running time over ngspatial. The poor performance of ngspatial comes from two reasons: (1) although ngspatial relies on



(a) Learning Epochs vs. Scalability

(b) Learning Epochs vs. Accuracy

Figure 4.8: Effect of Number of Learning Epochs on Scalability and Accuracy.

parallel processing in its sampling, prior estimation and parameters optimization steps, it runs a centralized approximate Bayesian inference algorithm [10]. In contrast, *TurboReg* is a fully distributed framework. (2) ngspatial requires estimating a prior distribution for each predictor variable, and hence it suffers from a huge latency before starting the actual learning process. Note that the ngspatial curve in Figure 4.7(a) is incomplete after a grid size of $3.5k$ cells because of a failure in satisfying the memory requirements needed for its internal computations. The running times of *TurboReg* and *G-TurboReg-8* are almost identical, except with grid sizes larger than $21k$ cells which have 13 seconds average difference. This shows that *TurboReg* is efficient when scaling up the grid size regardless of the neighbourhood degree.

Figure 4.7(b) shows the accuracy for each algorithm while using the same grid sizes in Figure 4.7(a). In this experiment, we divide the cells in each grid into training and testing sets, where we randomly select 15% of cells for testing and keep the rest for training. We repeat this process 5 times and then average the accuracy results (we follow the same approach in the whole accuracy experiments in our work). As can be seen in the figure, *G-TurboReg-8* has the same accuracy achieved by ngspatial, while basic *TurboReg* is at maximum 20% less accurate than both of them on average. The reason for that is the basic *TurboReg* captures less accurate neighbourhood dependencies than *G-TurboReg-8*. Note that the ngspatial curve is incomplete for grids with sizes more than $3.5k$ cells as in Figure 4.7(a).

Effect of Learning Epochs

In this section, we evaluate the performance, both scalability and accuracy, of basic *TurboReg* and *G-TurboReg-8* with ngspatial, while having four different values of learning epochs. In the following experiments, we fix the grid size to be $1k$ cells.

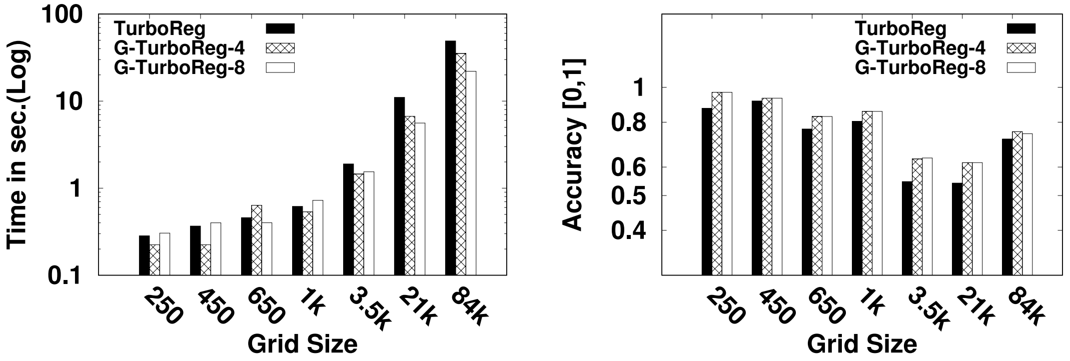
Figure 4.8(a) shows the running time for the different algorithms while changing the number of epochs from 100 to 100k. Both basic *TurboReg* and *G-TurboReg-8* significantly outperform ngspatial. They are at least 2 orders of magnitude faster than ngspatial. This is because of the parallel processing of learning epochs in *TurboReg* compared to the sequential learning in ngspatial. The results of ngspatial are incomplete after $10k$ epochs because its learning process requires saving huge intermediate state. In contrast, *TurboReg* never needs an intermediate state because it updates the model weights in place using the gradient descent optimization technique (See Algorithm 4). The figure also shows that the running times of *TurboReg* and *G-TurboReg-8* are identical and never depend on the neighbourhood degree. This confirms the complexity estimation of the weights learning algorithm in Section 4.7.

Figure 4.8(b) shows the accuracy of the different algorithms given the same setup in Figure 4.8(a). This experiment shows an interesting observation that both *TurboReg* and ngspatial can rapidly converge to their optimal values of weights (i.e., number of learning epochs less than 100). This is because ngspatial provides a good estimate to the prior of its predication and predictor variables, which makes the convergence process faster. In case of *TurboReg*, the rapid convergence happens because weights are shared among all locations which makes their values updated multiple times using the gradient descent optimization in each epoch. As a result, *TurboReg* just needs small number of epochs for weights convergence.

Effect of Neighbourhood Degree

In this section, we evaluate the performance, both scalability and accuracy, of basic *TurboReg* and two generalized variations *G-TurboReg-8* and *G-TurboReg-4*, while scaling up the grid size. Unlike *G-TurboReg-8*, *G-TurboReg-4* considers neighbourhood degree of 4, in which each location prediction depends on neighbours that share edges with this location only.

Figure 4.9(a) depicts the performance for each algorithm while using the same grid sizes in Figure 4.7(a). The results of this experiment confirm the previous ones we have shown in Figure 4.7(a). Increasing the neighbourhood degree leads to producing less number of predicates,



(a) Neighbourhood Degree vs. Scalability

(b) Neighbourhood Degree vs. Accuracy

Figure 4.9: Effect of Neighbourhood Degree on Scalability and Accuracy.

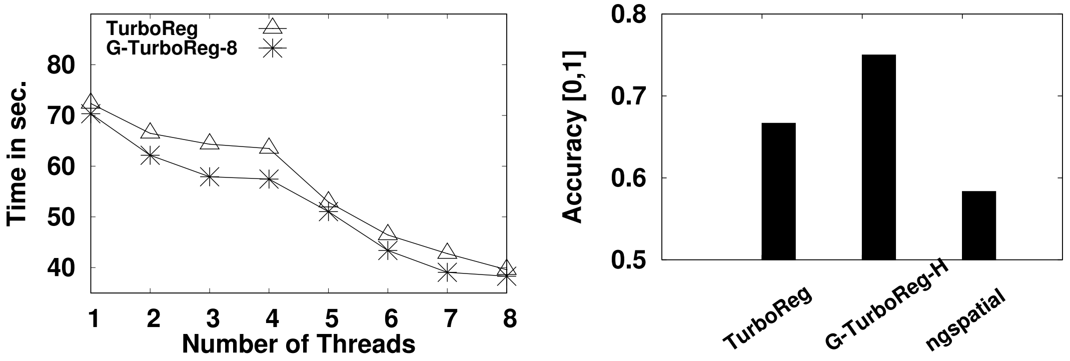
and hence less number of factor graph nodes to process, which makes the weights learning process faster. In this experiment, the performance of different algorithms are almost similar in case of small grid sizes (i.e., the average accuracy difference between the three algorithms is less than 0.1 seconds). However, the difference becomes significant in case of large grid sizes (average of 16 seconds difference for grid size of $84k$ cells).

Figure 4.9(b) shows the accuracy of *TurboReg*, *G-TurboReg-4* and *G-TurboReg-8* using the same setup of grid sizes. As we can see, the accuracy of *TurboReg* is 9% less accurate than both *G-TurboReg-4* and *G-TurboReg-8*. Note that *G-TurboReg-4* and *G-TurboReg-8* almost have the same accuracy. This is a spatial case for the Ebird dataset because we observe that the significant information between neighbourhoods with degrees 8 and 4 is very little, which makes the accuracy in the two cases are pretty similar.

Effect of Number of Threads and Hybrid Neighbourhood Degrees

Figure 4.10(a) shows the effect of increasing the number of threads from 1 to 8 on *TurboReg* and *G-TurboReg-8*. These threads are used to parallelize the work in the weights learner module of *TurboReg*. As expected, the performance of both algorithms linearly improves. For example, the running time of *TurboReg* using 8 threads is 2 times faster than using 1 thread. This shows the ability of autologistic to scale up with system threads.

Figure 4.10(b) shows the accuracy of both autologistic and ngspatial in case of having hybrid neighbourhood degrees (i.e., each location has a different neighbourhood degree). In this



(a) Number of Threads vs. Scalability

(b) Hybrid Niehgbourhood Degrees vs. Accuracy

Figure 4.10: Effect of Number of Threads on Scalability, and Hybrid Neighbourhood Degrees on Accuracy.

experiment, we use the MNCrimes dataset which consists of locations with 1 to 9 neighbourhood degrees. We compare the basic *TurboReg* that runs pairwise neighbouring depdencies, *G-TurboReg-H* that runs adaptive neighbourhood dependencies, and ngspatial. We find that the accuracy of *G-TurboReg-H* is higher than *TurboReg* with 12% and ngspatial with 29%.

4.9 Related Work

In this section, we first provide an overview of the existing theoretical and computational models of autologistic regression. Then, we briefly mention other related spatial regression models.

- **Autologistic Theoretical Models.** There are two main theoretical models of autologistic regression: (1) *Traditional model* [11] simply estimates the logistic function of the predication probability at any location as a linear combination of predictors at this location and the predictions of its neighbours. However, this model biases its prediction to the presence case (i.e., predicted value is 1) in case of sparse training data. (2) *Centered model* [49] is similar to the traditional model, however, the model parameters are normalized to avoid the biased cases. This adds more complexity when learning the model parameters. *TurboReg* is the first framework to implement those models on a large-scale without sacrificing the accuracy of learned model parameters. Recent research has proposed an extension for spatio-temporal autologistic models [54, 161] (and centered variants [53, 162]), which incorporates the temporal dependence between predictions at the same location. However, this line of research is out of the scope of this work.

- **Autologistic Computational Methods.** A wide array of techniques that are capable of learning the autologistic model parameters on a small scale (see [10, 163] for a comprehensive survey, and [51] for open-source implementations). Learning the autologistic model parameters is much harder than learning the parameters of classical non-spatial regression models due to the spatial dependence effect. Thus, the techniques are categorized into three main categories based on their methods of approximation to the original parameters distributions: Pseudo likelihood estimation [48, 54](and centered variants [10]), Monte Carlo likelihood estimation [50](and centered variants [10, 54]), Bayesian inference estimation [52, 164] (and centered variants [51, 54]). *TurboReg*, conversely, is the first technique to apply large-scale Markov Chain Monte Carlo estimation to learn the autologistic model parameters.

- **Other Spatial Regression Models.** Autologistic models belong to the class of non-Gaussian spatial modelling [165], in which the spatial dependence between predictions is conditionally modelled through direct neighbours. However, there are three other classes: (1) linear spatial models [165], (2) spatial generalized linear models [166] and (3) Gaussian Markov random field models [167], that encode the spatial dependence through a distance-based covariance matrix. This matrix defines how much the prediction in one location is affected by predictions in all other locations based on their relative distances. Another main difference is that autologistic models focus on binary predictions, while other classes are mainly developed for continuous and categorical predictions.

4.10 Conclusions

This chapter introduced *TurboReg*, a scalable framework for building spatial logistic regression models (a.k.a autologistic models) to predict spatial binary data. *TurboReg* provides an efficient modeling for the autologistic regression problem using Markov Logic Networks (MLN), which is a scalable statistical learning framework. *TurboReg* employs first-order logic predicates, a spatially-partitioned factor graph data structure, and an efficient gradient descent-based optimization technique to learn the autologistic model parameters. Experimental analysis using real and synthetic data sets shows that *TurboReg* achieves at least three orders of magnitude performance gain over existing state-of-the-art techniques while preserving the same accuracy.

Chapter 5

RegRocket: Scalable Multinomial Autologistic Regression Using Markov Logic Networks

5.1 Introduction

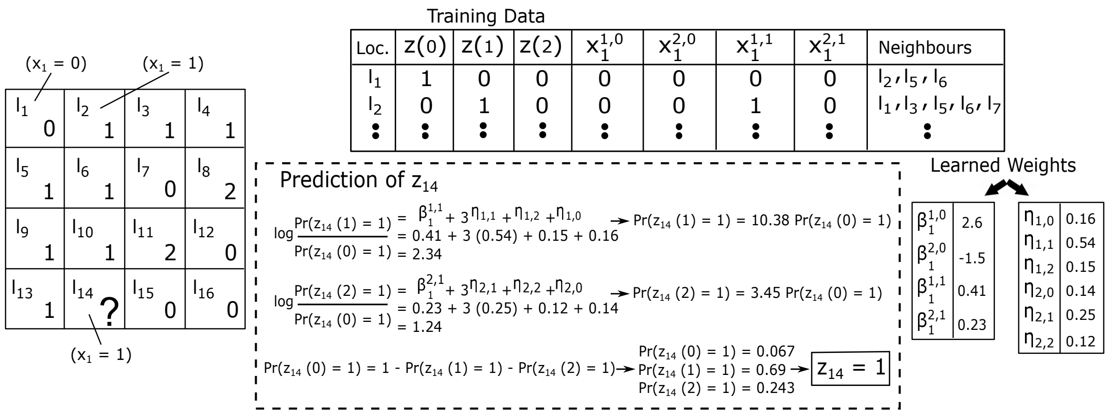
Autologistic regression [10, 11, 49] is an important statistical tool for predicting and analysing spatial phenomena in many scientific domains (e.g., Earth observations [140, 141], Epidemiology [142–144], Ecology [145, 146], Agriculture [147, 148], Archeology [168] and Management [149, 164]). Unlike standard logistic regression [151, 152] that assumes predictions of spatial phenomena over neighbouring locations are completely independent of each other, autologistic regression (See Section 4.2) takes into account the spatial dependence between neighbouring locations while building and running the prediction model (i.e., neighbouring locations tend to systematically affect each other [95]).

However, myriad applications require the autologistic regression model to be built over large *multinomial* (i.e., categorical) spatial data. Examples of these applications include multinomial brain [169] and satellite images [170] analysis. In these applications, the prediction and/or predictor variables in the regression model are multinomial, which means that the value of any variable comes from a set of possible values (i.e., domain values). However, existing methods for autologistic regression (e.g., see [10, 48, 52, 53]) are specifically designed for autologistic models with *binary* prediction and predictor variables (i.e., each variable takes either 0 or 1) only, and hence are not applicable for the multinomial case [163].

In this chapter, we introduce *RegRocket*; the first scalable framework for building autologistic models with *multinomial* prediction and predictor variables. *RegRocket* does not need to sample training data sets. It can support the prediction over grids of 1 million cells in few minutes. *RegRocket* is the successor of *TurboReg* [61] (See Chapter 4), from which it is distinguished by: (1) Providing a new MLN representation along with its theoretical foundation for multinomial autologistic models, unlike *TurboReg* that considers binary autologistic regression models only. The MLN representation of *RegRocket* can be considered as a generalization of its counterpart in *TurboReg*. (2) Adapting the weights learner module of *TurboReg* to efficiently implement the new MLN representation of the multinomial case. (3) Providing experimental study of the different system settings in terms of running time, and prediction accuracy while employing the MLN-based multinomial autologistic models.

We experimentally evaluate *RegRocket* using two real datasets of the daily distribution of bird species [150], and the land cover distribution of Minnesota, USA [171, 172]. We compare the accuracy and scalability of the built autologistic regression models over each dataset using the basic *RegRocket* and two generalized variations of *RegRocket*, that consider higher neighbouring interactions between predictions, with a state-of-the-art open-source autologistic model computational method, namely ngspatial [51]. Our experimental evaluation shows that *RegRocket* is scalable to large-scale autologistic models, while achieving a high-level of accuracy in estimating the model parameters.

The rest of this chapter is organized as follows: Section 5.2 gives a brief background of multinomial autologistic models. Section 5.3 describes how multinomial autologistic regression is modeled using MLN. Section 5.4 illustrates the details of the weights learning module for the categorical (i.e., multinomial) case. Section 5.5 provides the experimental analysis of *RegRocket*. Finally, Section 5.6 concludes the chapter.



l_1 0	l_2 1	l_3 1	l_4 1
l_5 1	l_6 1	l_7 0	l_8 2
l_9 1	l_{10} 1	l_{11} 2	l_{12} 0
l_{13} 1	l_{14} ?	l_{15} 0	l_{16} 0

Training Data

Loc.	z(0)	z(1)	z(2)	$x_1^{1,0}$	$x_1^{2,0}$	$x_1^{1,1}$	$x_1^{2,1}$	Neighbours
l_1	1	0	0	0	0	0	0	l_2, l_5, l_6
l_2	0	1	0	0	0	1	0	l_1, l_3, l_5, l_6, l_7
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Prediction of z_{14}

$$\log\frac{Pr(z_{14}(1)=1)}{Pr(z_{14}(0)=1)} = \beta_1^{1,1} + 3\eta_{1,1} + \eta_{1,2} + \eta_{1,0} = 0.41 + 3\,(0.54) + 0.15 + 0.16 = 2.34 \rightarrow Pr(z_{14}(1)=1) = 10.38\,Pr(z_{14}(0)=1)$$

$$\log\frac{Pr(z_{14}(2)=1)}{Pr(z_{14}(0)=1)} = \beta_1^{2,1} + 3\eta_{2,1} + \eta_{2,2} + \eta_{2,0} = 0.23 + 3\,(0.25) + 0.12 + 0.14 = 1.24 \rightarrow Pr(z_{14}(2)=1) = 3.45\,Pr(z_{14}(0)=1)$$

$$Pr(z_{14}(0)=1) = 1 - Pr(z_{14}(1)=1) - Pr(z_{14}(2)=1) \rightarrow \begin{matrix} Pr(z_{14}(0)=1) = 0.067 \\ Pr(z_{14}(1)=1) = 0.69 \\ Pr(z_{14}(2)=1) = 0.243 \end{matrix} \rightarrow \boxed{z_{14} = 1}$$

Learned Weights

$\beta_1^{1,0}$	2.6
$\beta_1^{2,0}$	-1.5
$\beta_1^{1,1}$	0.41
$\beta_1^{2,1}$	0.23

$\eta_{1,0}$	0.16
$\eta_{1,1}$	0.54
$\eta_{1,2}$	0.15
$\eta_{2,0}$	0.14
$\eta_{2,1}$	0.25
$\eta_{2,2}$	0.12

Figure 5.1: An Example on Multinomial Autologistic Regression.

5.2 Preliminaries of Multinomial Autologistic Regression

Binary autologistic models (See Section 4.2) can be extended to the case of multinomial (i.e., categorical) predictions and predictors. In this case, each prediction variable z_i has r possible outcomes $\mathcal{D}_{z_i} = \{\lambda_1, \lambda_2, ..., \lambda_r\}$, and each predictor variable $x_j(i)$ has q possible domain values $\mathcal{D}_{x_j(i)} = \{t_1, t_2, ..., t_q\}$. Since the variables are not binary, the model specified in Equation 4.1 is no longer valid for the multinomial case. Our approach to obtain the appropriate model for a prediction variable with r possible outcomes is to build $r - 1$ independent binary models, in which one outcome is chosen as a *pivot* and then the other $r - 1$ outcomes are separately regressed against the pivot outcome. Eventually, the probability of predicting the pivot outcome is calculated based on these built $r - 1$ binary models (i.e., $1 - \sum_{\lambda \neq p} Pr(outcome\ is\ \lambda)$ where p is the pivot outcome). Such approach is already implemented in classical multinomial logistic regression [173–175], yet, without considering the spatial dependence (i.e., neighbour-based regression terms).

In the generated binary models, each multinomial prediction variable z_i will be represented with r binary random variables $\{z_i(\lambda) \in \{0,1\} \mid \lambda \in \mathcal{D}_{z_i}\}$, where $z_i(\lambda)$ indicates whether the prediction value at location l_i is λ or not. In addition, each multinomial predictor $x_j(i)$ will be represented as a set of $(r-1)q$ binary random variables $\{x_j^{\lambda,t}(i) \mid \lambda \in \mathcal{D}_{z_i} - \{p\}, t \in \mathcal{D}_{x_j(i)}\}$, where λ is a non-pivot outcome to be predicted at location l_i and t is a possible domain value of $x_j(i)$. The variable $x_j^{\lambda,t}(i)$ represents a binary predictor (i.e., $\{x_j^{\lambda,t}(i) \in \{0,1\}\}$) in the autologistic regression model that is built for the binary prediction variable $z_i(\lambda)$. Assuming the pivot outcome of prediction variable z_i is λ_r, the $r-1$ conditional probabilities corresponding to z_i given the values of current predictor variables $\mathcal{X}$ and the neighbouring prediction variables $\mathcal{Z}_{\mathcal{N}_i}$ can be estimated as follows:

$$\begin{cases} \log \frac{Pr(z_i(\lambda_1)=1|\mathcal{X}(i),\mathcal{Z}_{\mathcal{N}_i})}{Pr(z_i(\lambda_r)=1|\mathcal{X}(i),\mathcal{Z}_{\mathcal{N}_i})} = \sum_{j=1}^{m} \sum_{t\in\mathcal{D}_{x_j(i)}} \beta_j^{\lambda_1,t} x_j^{\lambda_1,t}(i) + \sum_{k\in\mathcal{N}_i} \sum_{s\in\mathcal{D}_{z_k}} \eta_{\lambda_1,s} z_k(s) \\ \log \frac{Pr(z_i(\lambda_2)=1|\mathcal{X}(i),\mathcal{Z}_{\mathcal{N}_i})}{Pr(z_i(\lambda_r)=1|\mathcal{X}(i),\mathcal{Z}_{\mathcal{N}_i})} = \sum_{j=1}^{m} \sum_{t\in\mathcal{D}_{x_j(i)}} \beta_j^{\lambda_2,t} x_j^{\lambda_2,t}(i) + \sum_{k\in\mathcal{N}_i} \sum_{s\in\mathcal{D}_{z_k}} \eta_{\lambda_2,s} z_k(s) \\ ... \\ \log \frac{Pr(z_i(\lambda_{r-1})=1|\mathcal{X}(i),\mathcal{Z}_{\mathcal{N}_i})}{Pr(z_i(\lambda_r)=1|\mathcal{X}(i),\mathcal{Z}_{\mathcal{N}_i})} = \sum_{j=1}^{m} \sum_{t\in\mathcal{D}_{x_j(i)}} \beta_j^{\lambda_{r-1},t} x_j^{\lambda_{r-1},t}(i) + \sum_{k\in\mathcal{N}_i} \sum_{s\in\mathcal{D}_{z_k}} \eta_{\lambda_{r-1},s} z_k(s) \end{cases} \tag{5.1}$$

As shown in Equation 5.1, each binary predictor $x_j^{\lambda,t}(i)$ is associated with one weight $\beta_j^{\lambda,t}$. Moreover, in contrast to Equation 4.1 which has one weight η for the whole *neighbour-based* regression terms, the multinomial autologistic model defines a weight $\eta_{\lambda,s}$ for each possible pair of variables $(z_i(\lambda), z_k(s))$, where $z_i(\lambda)$ and $z_k(s)$ correspond to the predictions of a non-pivot outcome λ at location l_i (i.e., $\lambda \in \mathcal{D}_{z_i} - \{p\}$) and any outcome s at each neighbouring location $k \in \mathcal{N}_i$ (i.e., $s \in \mathcal{D}_{z_k}$). The main reason for having multiple η weights is to capture more precise spatial dependencies among neighbouring predictions compared to the one weight in traditional binary models. The objective of our work is to build multinomial autologistic models that achieve both high prediction accuracy and low running time by learning the values of model parameters $\theta = \{\beta, \eta\}$, where $\beta = \{\beta_1^{\lambda_1,t_1}, ..., \beta_m^{\lambda_{r-1},t_q}\}$ and $\eta = \{\eta_{\lambda_1,\lambda_1}, ..., \eta_{\lambda_{r-1},\lambda_r}\}$, from previous observations (i.e., training data) of predictions and predictors at locations $\mathcal{L}$, in a scalable and efficient manner. Note that the total number of weights to be learned in β and η are $mq(r-1)$ and $r(r-1)$, respectively.

Assumptions. In general, prediction and predictor variables can be either binary, multinomial, or continuous. However, we focus only on multinomial variables. The binary case has been discussed in Chapter 4, and the extension to continuous case is out of scope of this thesis.

Example. Figure 5.1 shows a numerical example of multinomial autologistic regression. In

Predicate	Weight
f_1: 1	0
f_2: $z_1(0) \wedge z_2(1)$	$\eta_{2,1}$
f_3: $z_1(0) \wedge z_2(2)$	$\eta_{2,0}$
f_4: $z_1(1) \wedge x_1^{1,0}(1)$	$\beta_1^{1,0}$
f_5: $z_1(1) \wedge x_1^{1,1}(1)$	$\beta_1^{1,1}$
f_6: $z_1(1) \wedge z_2(0)$	$\eta_{1,0}$
f_7: $z_1(1) \wedge z_2(1)$	$\eta_{1,1}$
f_8: $z_1(1) \wedge z_2(2)$	$\eta_{1,2}$
f_9: $z_1(2) \wedge x_1^{2,0}(1)$	$\beta_1^{2,0}$
f_{10}: $z_1(2) \wedge x_1^{2,1}(1)$	$\beta_1^{2,1}$
f_{11}: $z_1(2) \wedge z_2(0)$	$\eta_{2,0}$
f_{12}: $z_1(2) \wedge z_2(1)$	$\eta_{2,1}$
f_{13}: $z_1(2) \wedge z_2(2)$	$\eta_{2,2}$
f_{14}: 1	0
f_{15}: $z_2(1) \wedge x_1^{1,0}(2)$	$\beta_1^{1,0}$
f_{16}: $z_2(1) \wedge x_1^{1,1}(2)$	$\beta_1^{1,1}$
f_{17}: $z_2(2) \wedge x_1^{2,0}(2)$	$\beta_1^{2,0}$
f_{18}: $z_2(2) \wedge x_1^{2,1}(2)$	$\beta_1^{2,1}$

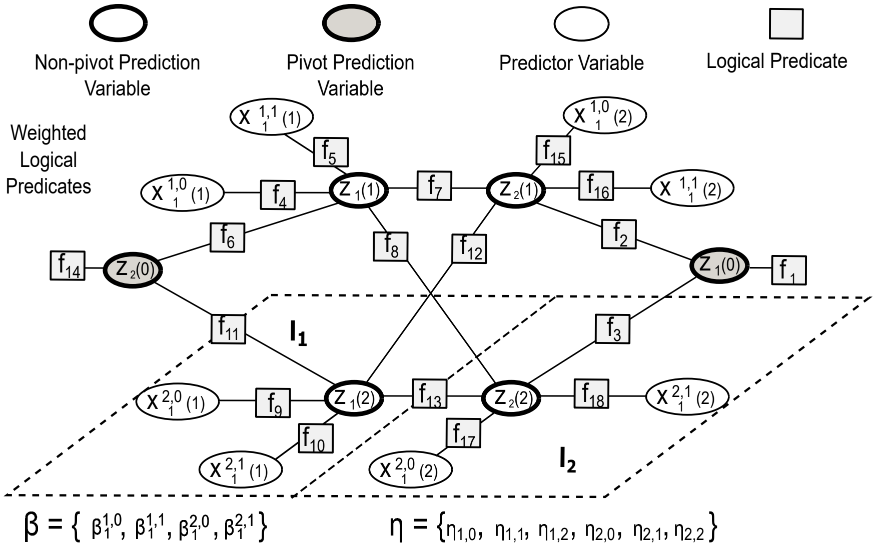


Figure 5.2: An Equivalent MLN Representation to A Multinomial Autologistic Model (logical predicates and their factor graph).

this example, we have a 4 x 4 grid (i.e., 16 cells), where each cell l_i has a prediction variable z_i with three possible outcomes (i.e., $\mathcal{D}_{z_i} = \{0, 1, 2\}$), and one binary predictor variable $x_1(i)$ (i.e., $\mathcal{D}_{x_1(i)} = \{0, 1\}$). Assuming the prediction outcome 0 as pivot, each location i has 3 binary prediction variables $\{z_i(0), z_i(1), z_i(2)\}$, and 4 binary predictor variables $\{x_1^{1,0}(i), x_1^{2,0}(i), x_1^{1,1}(i), x_1^{2,1}(i)\}$. As a result, we have 4 predictor-based and 6 neighbour-based weights. These weights are trained by the observations from all locations except l_{14} which is unknown (i.e., needs to be predicted). The example also shows the calculations to predict the value of z_{14} using the learned regression parameters. The probabilities of the three possible outcomes of z_i are first calculated, and then the outcome corresponding to the highest probability is selected as the prediction value.

5.3 Multinomial Autologistic Regression via Markov Logic Networks

In this section, we describe how the MLN framework is exploited to efficiently solve the multinomial autologistic regression problem. In particular, we discuss the MLN-based model for the basic multinomial autologistic regression, along with its theoretical foundation (Section 5.3.1).

To represent a multinomial autologistic model using MLN, *RegRocket* extends the MLN-based binary autologistic model in *TurboReg* [61] (Section 4.3.1) to support the multinomial case. *TurboReg* represents all binary autologistic regression terms, whether predictor-based or neighbour-based, as a set of weighted bitwise-AND logical predicates (i.e., weighted MLN constraints). In *RegRocket*, we follow the same approach of mapping from regression terms to logical predicates, however, with two main modifications. The first modification is to apply this mapping on each regression term defined in the $r - 1$ binary regression models of the multinomial case (Equation 5.1). For each prediction variable $z_i(\lambda)$ corresponding to a non-pivot possible outcome λ at location l_i, each *predictor-based* regression term $\beta_j^{\lambda,t} x_j^{\lambda,t}(i)$ has an equivalent bitwise-AND predicate defined over $z_i(\lambda)$ and $x_j^{\lambda,t}(i)$ (i.e., $z_i(\lambda) \wedge x_j^{\lambda,t}(i)$) with weight $\beta_j^{\lambda,t}$. Similarly, each *neighbour-based* regression term $\eta_{\lambda,s} z_k(s)$ has an equivalent bitwise-AND predicate defined over $z_i(\lambda)$ and $z_k(s)$ with weight $\eta_{\lambda,s}$. Recall that all prediction and predictor variables in Equation 5.1 are binary, and hence, it is valid to provide equivalent logical predicates to them. The second modification is to define a constant predicate of value 1 and weight 0 for any prediction variable $z_i(p)$ corresponding to a pivot outcome p at location l_i. The theoretical foundation of the proposed MLN-based multinomial autologistic model is described in Section 5.3.1. Note that, using the proposed model, the autologistic regression parameters $\theta = \{\beta, \eta\}$ are translated into a set of weights $\mathcal{W}$ of MLN constraints (i.e., proposed equivalent bitwise-AND predicates), and hence learning the autologistic model parameters θ becomes equivalent to learning the values of $\mathcal{W}$ in MLN (See Section 2.2).

Example. Figure 5.2 shows an example of translating a multinomial autologistic regression model into an equivalent MLN. This example defines a model with multinomial prediction of 3 possible outcomes $\{0, 1, 2\}$ (i.e., three binary prediction variables $\{z_i(0), z_i(1), z_i(2)\}$ at each location l_i) where the pivot outcome is 0, and one multinomial predictor of 2 domain values $\{0, 1\}$ (i.e., four binary predictor variables $\{x_1^{1,0}(i), x_1^{2,0}(i), x_1^{1,1}(i), x_1^{2,1}(i)\}$ at each location l_i). The model is built for two neighbouring locations $\{l_1, l_2\}$. We first translate the autologistic model into a set of 16 bitwise-AND predicates and 2 constant predicates along with 4 predictor-based weights $\beta_1 = \{\beta_1^{1,0}, \beta_1^{2,0}, \beta_1^{1,1}, \beta_1^{2,1}\}$ and 6 neighbour-based weights $\eta = \{\eta_{1,0}, \eta_{1,1}, \eta_{1,2}, \eta_{2,0}, \eta_{2,1}, \eta_{2,2}\}$. Then, these predicates are translated into a factor graph which can be used to learn the weights β_1 and η. Note that duplicate predicates that come from neighbouring variables are removed to avoid redundancy (e.g., the neighbouring variables

$z_1(2)$ and $z_2(2)$ have two equivalent $z_1(2) \wedge z_2(2)$ and $z_2(2) \wedge z_1(2)$ neighbour-based predicates, respectively, however, we keep only one of them). Note that we follow the generalization approach discussed in (Section 4.3.3) to extend this MLN-based model in case of more complicated multinomial autologistic regression scenarios.

5.3.1 Theoretical Foundation of RegRocket using MLN

Theorem 2 *Given an autologistic model defined over n locations $\mathcal{L} = \{l_1, ..., l_n\}$ with*

- *n multinomial prediction variables $\mathcal{Z} = \{z_1, ..., z_n\}$, each has r possible outcomes $\mathcal{D}_{z_i} = \{\lambda_1, ..., \lambda_r\}$, and are represented with a set of nr binary variables $\{z_1(\lambda_1), ..., z_n(\lambda_r)\}$,*
- *m weighted multinomial predictor variables $\mathcal{X}(i) = \{x_1(i), ..., x_m(i)\}$ at each location i, each has q possible domain values $\mathcal{D}_{x_j(i)} = \{t_1, ..., tq\}$, and are represented with a set of $mq(r-1)$ weighted binary variables $\{\beta_1^{\lambda_1,1} x_1^{\lambda_1,1}(i), ..., \beta_m^{\lambda_{r-1},q} x_m^{\lambda_{r-1},q}(i)\}$, where λ_r is a pivot outcome of any multinomial prediction z_i, and*
- *$r(r-1)$ neighbouring weights $\eta = \{\eta_{\lambda_1,\lambda_1}, ..., \eta_{\lambda_{r-1},\lambda_r}\}$*

there is an equivalent Markov Logic Network (MLN) to this autologistic model, if and only if:

- *each predictor-based regression term $\beta_j^{\lambda,t} x_j^{\lambda,t}(i)$ at location l_i has an equivalent bitwise-AND predicate $z_i(\lambda) \wedge x_j^{\lambda,t}(i)$ with weight $\beta_j^{\lambda,t}$, where λ is not a pivot outcome (i.e., $\lambda \neq \lambda_r$).*
- *each neighbour-based regression term $\eta_{\lambda,s} z_k(s)$ at location l_i has an equivalent bitwise-AND predicate $z_i(\lambda) \wedge z_k(s)$ with weight $\eta_{\lambda,s}$, where λ is not a pivot outcome (i.e., $\lambda \neq \lambda_r$).*
- *each prediction variable $z_i(\lambda_r)$ at location l_i is associated with a constant predicate of value 1 and weight 0.*

Proof. Based on the conditional probabilities of multinomial autologistic model in Equation 5.1, the probability of predicting any possible non-pivot outcome λ (i.e., $\lambda \neq \lambda_r$) at location l_i given the predictor variables $\mathcal{X}(i)$ at l_i and the neighbouring prediction variables $\mathcal{Z}_{\mathcal{N}_i}$ can be estimated as shown in following equation:

$$Pr(z_i(\lambda) = 1 \mid \phi_i) = Pr(z_i(\lambda_r) = 1 \mid \phi_i) \exp \Big(\sum_{j=1}^{m} \sum_{t \in \mathcal{D}_{x_j(i)}} \beta_j^{\lambda,t} x_j^{\lambda,t}(i) + \sum_{k \in \mathcal{N}_i} \sum_{s \in \mathcal{D}_{z_k}} \eta_{\lambda,s} z_k(s) \Big) \tag{5.2}$$

where $\phi_i = \{\mathcal{X}(i), \mathcal{Z}_{\mathcal{N}_i}\}$. As a result, the probability of predicting the pivot outcome λ_r at location l_i can be calculated as follows:

$$\begin{aligned} Pr(z_i(\lambda_r) = 1 \mid \phi_i) &= 1 - \sum_{\lambda \in \mathcal{D}_{z_i}, \lambda \neq \lambda_r} Pr(z_i(\lambda) = 1 \mid \phi_i) \\ &= 1 - \sum_{\lambda \in \mathcal{D}_{z_i}, \lambda \neq \lambda_r} Pr(z_i(\lambda_r) = 1 \mid \phi_i) \exp \Big(\sum_{j=1}^{m} \sum_{t \in \mathcal{D}_{x_j(i)}} \beta_j^{\lambda,t} x_j^{\lambda,t}(i) + \sum_{k \in \mathcal{N}_i} \sum_{s \in \mathcal{D}_{z_k}} \eta_{\lambda,s} z_k(s) \Big) \\ &= \frac{1}{1 + \sum\limits_{\lambda \in \mathcal{D}_{z_i}, \lambda \neq \lambda_r} \exp \Big(\sum\limits_{j=1}^{m} \sum\limits_{t \in \mathcal{D}_{x_j(i)}} \beta_j^{\lambda,t} x_j^{\lambda,t}(i) + \sum\limits_{k \in \mathcal{N}_i} \sum\limits_{s \in \mathcal{D}_{z_k}} \eta_{\lambda,s} z_k(s) \Big)} \end{aligned} \tag{5.3}$$

From equations 5.2 and 5.3, the probability distribution of any outcome prediction $z_i(\lambda)$ is expressed as follows:

$$Pr(z_i(\lambda) = 1 \mid \phi_i) = \begin{cases} \dfrac{\exp \Big(\sum\limits_{j=1}^{m} \sum\limits_{t \in \mathcal{D}_{x_j(i)}} \beta_j^{\lambda,t} x_j^{\lambda,t}(i) + \sum\limits_{k \in \mathcal{N}_i} \sum\limits_{s \in \mathcal{D}_{z_k}} \eta_{\lambda,s} z_k(s) \Big)}{1 + \sum\limits_{h \in \mathcal{D}_{z_i}, h \neq \lambda_r} \exp \Big(\sum\limits_{j=1}^{m} \sum\limits_{t \in \mathcal{D}_{x_j(i)}} \beta_j^{h,t} x_j^{h,t}(i) + \sum\limits_{k \in \mathcal{N}_i} \sum\limits_{s \in \mathcal{D}_{z_k}} \eta_{h,s} z_k(s) \Big)} & \lambda \neq \lambda_r \\ \dfrac{1}{1 + \sum\limits_{h \in \mathcal{D}_{z_i}, h \neq \lambda_r} \exp \Big(\sum\limits_{j=1}^{m} \sum\limits_{t \in \mathcal{D}_{x_j(i)}} \beta_j^{h,t} x_j^{h,t}(i) + \sum\limits_{k \in \mathcal{N}_i} \sum\limits_{s \in \mathcal{D}_{z_k}} \eta_{h,s} z_k(s) \Big)} & \lambda = \lambda_r \end{cases} \tag{5.4}$$

Now, assume a model that consists of $nr + nmq(r-1)$ binary random variables $\mathcal{V} = \{\mathcal{Z}, \mathcal{X}\} = \{z_1(\lambda_1), ..., z_n(\lambda_r), x_1^{\lambda_1,1}(i), ..., x_m^{\lambda_{r-1},q}(i)\}$, coming from nr prediction and $nmq(r-1)$ predictor variables over all locations $\mathcal{L}$. In addition, in case $\lambda \neq \lambda_r$, assume a set of constraints $\mathcal{F} = \{F_1, ..., F_n\}$ are defined over variables $\mathcal{V}$, where constraints F_i at location l_i consist of two subsets of constraints:

- The first subset consists of $mq(r-1)$ bitwise-AND predicates $z_i(\lambda) \wedge x_j^{\lambda,t}(i)$ with β

weights (each predicate corresponds to a predictor-based regression term).

- The second subset consists of $r(r-1)s_i$ bitwise-AND predicates $z_i(\lambda) \wedge z_k(s)$ with η weights (each corresponds to a neighbour-based regression term) where s_i is the size of neighbouring locations $\mathcal{N}_i$ of location l_i.

Finally, in case $\lambda = \lambda_r$ at each location l_i, assume one constant predicate of value 1 with weight 0. Based on these assumptions, the model satisfies the two main properties in Section 2.3 that are needed to represent it using MLN, and hence its joint probability distribution over $\mathcal{V}$ is estimated based on Equation 2.7 as follows:

$$Pr(\mathcal{Z},\mathcal{X}) = \begin{cases} \frac{1}{C}\exp\Big(\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{m}\sum\limits_{(\lambda,t)\in\mathcal{D}_{z_i}\times\mathcal{D}_{x_j(i)}} \beta_j^{\lambda,t} f_{\lambda,t}(z_i(\lambda), x_j^{\lambda,t}(i)) \\ \qquad + \sum\limits_{i=1}^{n}\sum\limits_{k\in\mathcal{N}_i}\sum\limits_{(\lambda,s)\in\mathcal{D}_{z_i}\times\mathcal{D}_{z_k}} \eta_{\lambda,s} f_{\lambda,s}(z_i(\lambda), z_k(s))\Big) & \lambda \neq \lambda_r \\ \frac{1}{C} & \lambda = \lambda_r \end{cases} \tag{5.5}$$

where $f_{\lambda,t}(.)$ and $f_{\lambda,s}(.)$ represent functions to evaluate the bitwise-AND predicates $z_i(\lambda) \wedge x_j^{\lambda,t}(i)$ and $z_i(\lambda) \wedge z_k(s)$, respectively, and return either 1 or 0 as output. Note that in the case of pivot outcome λ_r, we have a constant predicate of value 1 with weight 0, and hence its probability becomes $\frac{1}{C}\exp(0(1)) = \frac{1}{C}$, where C is the normalization constant of the model.

Based on Equation 5.5, the conditional probability distribution of any prediction variable $z_i(\lambda)$ at location l_i given the predictor variables $\mathcal{X}(i)$ at l_i and its neighbouring prediction variables $\mathcal{Z}_{\mathcal{N}_i}$ (i.e., $\phi_i = \{\mathcal{X}(i), \mathcal{Z}_{\mathcal{N}_i}\}$) can be estimated as:

$$Pr(z_i(\lambda) = 1 \mid \phi_i) = \begin{cases} \frac{1}{C}\exp\Big(\sum\limits_{j=1}^{m}\sum\limits_{t\in\mathcal{D}_{x_j(i)}} \beta_j^{\lambda,t} f_{\lambda,t}(1, x_j^{\lambda,t}(i)) \\ \qquad + \sum\limits_{k\in\mathcal{N}_i}\sum\limits_{s\in\mathcal{D}_{z_k}} \eta_{\lambda,s} f_{\lambda,s}(1, z_k(s))\Big) & \lambda \neq \lambda_r \\ \frac{1}{C} & \lambda = \lambda_r \end{cases} \tag{5.6}$$

where the normalization constant C is calculated over all possible outcomes of z_i, to ensure the probability value of any outcome $\in [0,1]$, as follows:

$$C = \Big[\sum_{h \in \mathcal{D}_{z_i}, h \neq \lambda_r} \exp\Big(\sum_{j=1}^{m} \sum_{t \in \mathcal{D}_{x_j(i)}} \beta_j^{h,t} f_{h,t}(1, x_j^{h,t}(i)) + \sum_{k \in \mathcal{N}_i} \sum_{s \in \mathcal{D}_{z_k}} \eta_{h,s} f_{h,s}(1, z_k(s)) \Big) \Big] + \exp(0)$$

$$= 1 + \sum_{h \in \mathcal{D}_{z_i}, h \neq \lambda_r} \exp\Big(\sum_{j=1}^{m} \sum_{t \in \mathcal{D}_{x_j(i)}} \beta_j^{h,t} f_{h,t}(1, x_j^{h,t}(i)) + \sum_{k \in \mathcal{N}_i} \sum_{s \in \mathcal{D}_{z_k}} \eta_{h,s} f_{h,s}(1, z_k(s)) \Big)$$

Since all variables are binary, the evaluation of $f_{\lambda,t}$ and $f_{\lambda,s}$ can be represented as a mathematical multiplication (i.e., the value of $f_{\lambda,t}(1, x_j^{\lambda,t}(i))$ is $x_j^{\lambda,t}(i)$ and the value of $f_{\lambda,s}(1, z_k(s))$ is $z_k(s)$). As a result, the joint probability distribution of $z_i(\lambda)$ from Equation 5.6 becomes:

$$Pr(z_i(\lambda) = 1 \mid \phi_i) = \begin{cases} \dfrac{\exp\Big(\sum_{j=1}^{m} \sum_{t \in \mathcal{D}_{x_j(i)}} \beta_j^{\lambda,t} x_j^{\lambda,t}(i) + \sum_{k \in \mathcal{N}_i} \sum_{s \in \mathcal{D}_{z_k}} \eta_{\lambda,s} z_k(s) \Big)}{1 + \sum_{h \in \mathcal{D}_{z_i}, h \neq \lambda_r} \exp\Big(\sum_{j=1}^{m} \sum_{t \in \mathcal{D}_{x_j(i)}} \beta_j^{h,t} x_j^{h,t}(i) + \sum_{k \in \mathcal{N}_i} \sum_{s \in \mathcal{D}_{z_k}} \eta_{h,s} z_k(s) \Big)} & \lambda \neq \lambda_r \\ \dfrac{1}{1 + \sum_{h \in \mathcal{D}_{z_i}, h \neq \lambda_r} \exp\Big(\sum_{j=1}^{m} \sum_{t \in \mathcal{D}_{x_j(i)}} \beta_j^{h,t} x_j^{h,t}(i) + \sum_{k \in \mathcal{N}_i} \sum_{s \in \mathcal{D}_{z_k}} \eta_{h,s} z_k(s) \Big)} & \lambda = \lambda_r \end{cases} \tag{5.7}$$

which is the same probability distribution of the autologistic model defined in Equation 5.4. This means that the assumed model at the beginning, which can be represented with MLN, is equivalent to the multinomial autologistic model.

5.4 Weights Learner

This is the most important module in *RegRocket*, which takes the spatially-indexed factor graph, similar to *TurboReg* (See Section 4.6), along with learning configurations from the user and returns the final weights $\theta = \{\beta, \eta\}$ of the multinomial autologistic model.

Main Idea. A typical solution to efficiently learn the unknown weights $\mathcal{W}$ of any MLN model is to provide an approximate log-likelihood function for the full joint probability distribution in Equation 2.7 as follows [74]:

$$\log Pr(\mathcal{V} = v) = \sum_{i=1}^{h} w_i f_i(v) - \log C \tag{5.8}$$

Equation 5.8 provides an objective function to optimize when learning the weights $\mathcal{W}$ of the model. As shown in [74], we can estimate the gradient of any weight w_i as follows:

$$\frac{\partial}{\partial w_i} \log Pr(\mathcal{V} = v) = f_i(v) - E[f_i(v)] \tag{5.9}$$

where $E[f_i(v)]$ is the expected value of whether the i-th constraint (i.e., predicate) is satisfied or not. Equation 5.9 can direct how to incrementally converge to the weights that maximize the satisfaction of the MLN model. For example, by applying Equation 5.9 on the training data, if the gradient value of weight w_i is positive, then the current assignment of variables in $f_i(v)$ increases the satisfaction of the MLN model, and hence the corresponding weight w_i should be rewarded (i.e., should be increased), otherwise it should be punished (i.e., should be decreased). However, estimating the value of $E[f_i(v)]$ is known to be computationally-expensive in MLN models and requires approximate inference algorithms [74, 176].

As a result, instead of contrasting the satisfied value of the i-th predicate $f_i(v)$ against its expectation value $E[f_i(v)]$ (Equation 5.9), *RegRocket* contrasts the estimated prediction value of the autologistic model (Equation 5.1) where this predicate belongs to against the corresponding observed prediction from the training data. This approximation has been shown in a recent MLN-based application [2] to converge to the weights that maximize the satisfaction of the MLN model as long as there is no predicate whose observed value is unknown in the training data, which is the case of our regression models. In addition, this approximation is efficient-to-compute as the prediction is estimated by a direct substitution in Equation 5.1 (i.e., no need for approximate inference algorithms). As an example, to estimate the gradient value of the weight $\beta_j^{\lambda,t}$ of the predicate $z_i(\lambda) \wedge x_j^{\lambda,t}(i)$, *RegRocket* uses the following two items: (1) the estimated prediction value $\hat{z}_i(\lambda_v)$ based on the current value of $x_j^{\lambda,t}(i)$, and (2) the observed value of $z_i(\lambda_v)$ from the training data. If the estimated prediction $\hat{z}_i(\lambda_v)$ is similar to the observed prediction $z_i(\lambda_v)$, then the weight $\beta_j^{\lambda,t}$ should be rewarded, otherwise it should be punished.

To that end, we adapt a variation of the weights learner in *TurboReg* (Section 4.7) that punishes and rewards the weights of the MLN-based multinomial autologistic regression. Specifically, we provide a more efficient UPDATEWEIGHTS algorithm for the categorical case as described below. Note that we use the same LEARNWEIGHTS algorithm (Algorithm 3) that has been shown in Section 4.7.

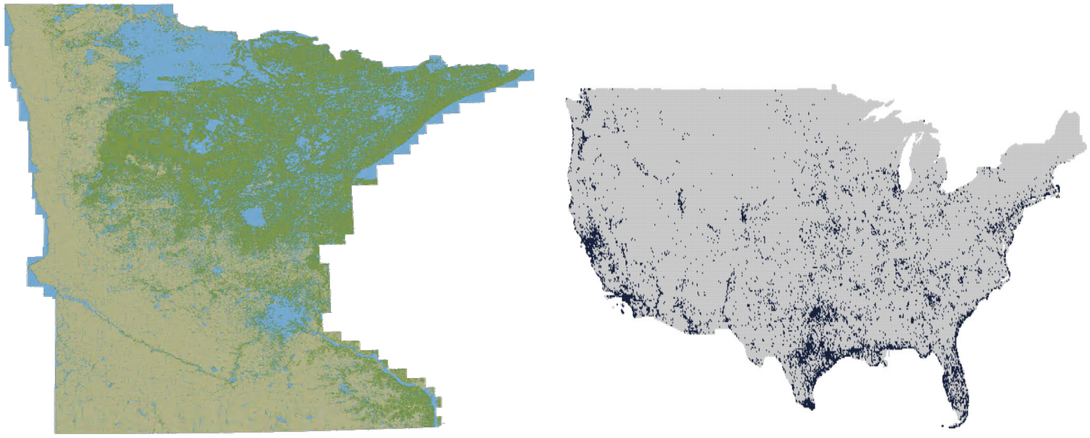
UPDATEWEIGHTS Algorithm. Algorithm 5 gives the pseudo code for our weights optimizer that applies the gradient descent optimization [75] technique to incrementally update the values of weights given a certain variable v (either non-pivot prediction or predictor) from the training data. Assume the outcome that appears in v is λ_v. Similarly, in case v is a predictor variable, assume the possible domain value in v is t_v. The main idea is to punish or reward current weights based on their performance in correctly estimating the prediction value $z_i(\lambda_v)$ at location l_i where the variable v belongs to.

The algorithm takes the following inputs: a variable v that belongs to the training data, the variables $\mathcal{V}$ and predicates $\mathcal{P}$ indexes in the grid cell containing v (i.e., graph index), and a step size α that controls the amount of punishing/rewarding during the optimization process. The algorithm keeps track of the current status of whether weights need to be punished or rewarded through a variable g, where it takes either 1 in case of rewarding or -1 in case of punishing, and is initialized by 1. The algorithm starts by estimating the prediction $\hat{z_i}(\lambda_v)$ at location l_i that contains v based on the current values of β and η using Equation 5.1. If the estimation $\hat{z_i}(\lambda_v)$ never matches the observed prediction value from the training data, then we set the status g to -1 (i.e., the associated weight with current variable v needs to be punished), otherwise the status remains rewarding. In case v is a predictor variable $x_j^{\lambda_v,t_v}(i)$, we only update its associated weight $\beta_j^{\lambda_v,t_v}$ by evaluating the gradient descent equation using the current values of g and α (Line 8 in Algorithm 5) and jump to the end of algorithm. In case v is the prediction variable $z_i(\lambda_v)$ itself, we apply the gradient descent optimization on all weights β associated with its predictors (Lines 10 to 12 in Algorithm 5), and on all weights η associated with the neighbouring predicates (Lines 15 to 25 in Algorithm 5) as well.

Complexity. The complexity of the two aforementioned algorithms can be estimated as

$$O(\frac{E}{S}\frac{(n^2r^2 + nrmq(r-1))}{C})$$

where nr is the number of prediction variables, $mq(r-1)$ is the number of predictor variables, C is the number of factor graph partitions, E is the number of learning epochs and S is the number of learning instances. This complexity can be further approximated to be $O(\frac{E}{S}\frac{(n^2r^2)}{C})$. Note that we assume having SC working threads to process C factor graph partitions in each of the S learning instances in parallel.



(a) MNLandCover Dataset

(b) Ebird Dataset

Figure 5.3: Datasets Used in *RegRocket* Experiments.

5.5 Experiments

In this section, we experimentally evaluate the accuracy and scalability of *RegRocket* in building multinomial autologistic models (i.e., learning their weights). To the best of our knowledge, *RegRocket* is the first end-to-end system that supports multinomial autologistic regression. As a result, we compare the performance of *RegRocket* with multinomial models built on top of a state-of-the-art binary autologistic regression package, namely ngspatial [51]. Specifically, we compare our performance with multinomial models built on top of the most accurate algorithm in ngspatial that employs Bayesian inference using Markov Chain Monte Carlo (MCMC) [10]. In addition, we extensively investigate the performance of different variations of *RegRocket* under different parameters including grid size, learning epoch, optimization step size, factor graph partitioning and parallelism.

5.5.1 Experimental Setup

Datasets

All experiments in this section are based on the following two grid datasets:

- *MNLandCover* dataset, which represents the land cover distribution in Minnesota state and is compiled from the USGS National Land Cover [171] and Multi-Resolution Land

Cover Consortium [172] data repositories. Figure 5.3(a) depicts the land cover distribution in Minnesota, where each grid cell is either crops (yellow color), forest (green color) or others (blue color). Thus, we generate a multinomial (i.e., categorical) prediction variable at each grid cell, where each variable takes one of these three possible values. As shown in a recent study [177], the land cover prediction is influenced by three factors; the elevation and slope of the ground as well as the distance to nearby roads. Based on this study, we also generate three multinomial predictors corresponding to these factors based on the elevation [178] and transportation [179] datasets of Minnesota at each grid cell. Each predictor takes one value out of 11 possible values. We generate six versions of this dataset with different grid sizes, ranging from 1000 to 1 million cells, to be used during most of our experiments.

- *Ebird* dataset [150], which is a real dataset of the daily distribution of a certain bird species, namely Barn Swallow, over North America. Each grid cell holds a predication of the bird existence in the cell or not (i.e., binary prediction). Figure 5.3(b) shows the Ebird data distribution, where blue dots refer to cells with bird existence. We generate six versions of this dataset with different grid sizes, ranging from 250 to 84000 cells, to be used during most of our experiments. This dataset has three multinomial predictors at each grid cell including bird observers, observing duration, and the spatial area covered by observers. Each predictor takes one value out of 3 possible values.

Parameters

Unless otherwise mentioned, Table 5.1 shows the settings of both MNLandCover and Ebird datasets. We select the $250k$ and $84k$ variations of MNLandCover and Ebird, receptively, to be used by default. In each dataset, we divide the cells in each grid into training and testing sets, where we randomly select 20% of cells for testing and keep the rest for training. All training and testing cells have ground truth predictions (i.e., no missing values). During the testing phase, we use the following three inputs to perform the prediction at any testing cell: (1) the learned regression model parameters, (2) the values of the predictors at this cell, and (3) the ground truth predictions at the neighbours of this cell.

Table 5.2 also shows the default learning configurations that are used with *RegRocket*. In most of the experiments, we run three variations of our system: the basic *RegRocket* that has pairwise neighbourhood relationships (i.e., neighbourhood degree of 1), and other two generalized variations with 4-ary and 8-ary neighbourhood relationships (i.e., neighbourhood degrees of 4 and 8), referred to as *RegRocket-4* and *RegRocket-8*, respectively (See Section 4.3.3). In *RegRocket-4*, each predication has a bitwise-AND predicate over the vertical and horizontal neighbours surrounding it (i.e., neighbours that share edges with this location only). Similarly, in *RegRocket-8*, each predication has a bitwise-AND predicate over the whole 8 neighbours (i.e., neighbours that share points with this location). Note that the neighbourhood relationships are pre-specified and fixed during both the training and testing phases.

Tables 5.3 and 5.4 show the total number of predicates that are generated for both datasets when using the basic *RegRocket*, *RegRocket-4* and *RegRocket-8* during our experiments.

Environment

We run all experiments on a single machine with Ubuntu Linux 14.04, 8 quad-core 3.00 GHz processors, 64GB RAM, and 4TB hard disk.

Metrics

We use the total running time of learning weights as a scalability evaluation metric. To measure the model accuracy, we use the following three metrics to evaluate the prediction quality of each outcome λ (i.e., category):

- **Precision (Prec.):** the number of *correctly* predicted cells with the outcome λ over the total number of predicted cells with the outcome λ.
- **Recall (Rec.):** the number of *correctly* predicted cells with the outcome λ over the total number of testing cells that are actually labelled with λ from the ground truth.
- **F1-score (F1):** the harmonic mean of precision and recall for the outcome λ as:

$$2 \times \frac{(Prec. \times Rec.)}{Prec. + Rec.}$$

To handle the multinomial case, we calculate these three metrics for each outcome, and then report the average of each metric over the total number of outcomes.

Parameter	MNLandCover Dataset	Ebird Dataset
Grid Training Size	200000 cells	67200 cells
Grid Testing Size	50000 cells	16800 cells
Number of Predictors	3	3
Number of Possible Predictor Values	11	3
Number of Possible Prediction Values	3	2

Table 5.1: Default Dataset-specific Parameters.

Parameter	Default Value
Learning Epochs E	1000
Neighbourhood Degree D	1, 4, 8
Step Size α	0.001
Number of Threads	7
Factor Graph Partitions C	200

Table 5.2: Default Learning-specific Parameters.

Grid Size	*RegRocket*	*RegRocket-4*	*RegRocket-8*
$1k$	$70k$	$66.8k$	$66.7k$
$4k$	$280k$	$267.7k$	$267.4k$
$15k$	$1.05m$	$1m$	$998.5k$
$60k$	$4.2m$	$4m$	$3.9m$
$250k$	$17.5m$	$16.7m$	$16.5m$
$1m$	$70m$	$66.9m$	$66.7m$

Table 5.3: Number of Predicates for the MNLandCover dataset.

Grid Size	*RegRocket*	*RegRocket-4*	*RegRocket-8*
250	$3.2k$	$2.4k$	$2.3k$
$1k$	$13k$	$9.8k$	$9.7k$
$3.5k$	$45.5k$	$34.7k$	$34.5k$
$5k$	$65k$	$49.7k$	$49.4k$
$21k$	$273k$	$209k$	$208.7k$
$84k$	$1.09m$	$838.8k$	$837.6k$

Table 5.4: Number of Predicates for the Ebird dataset.

5.5.2 Experimental Results

Effect of Grid Size

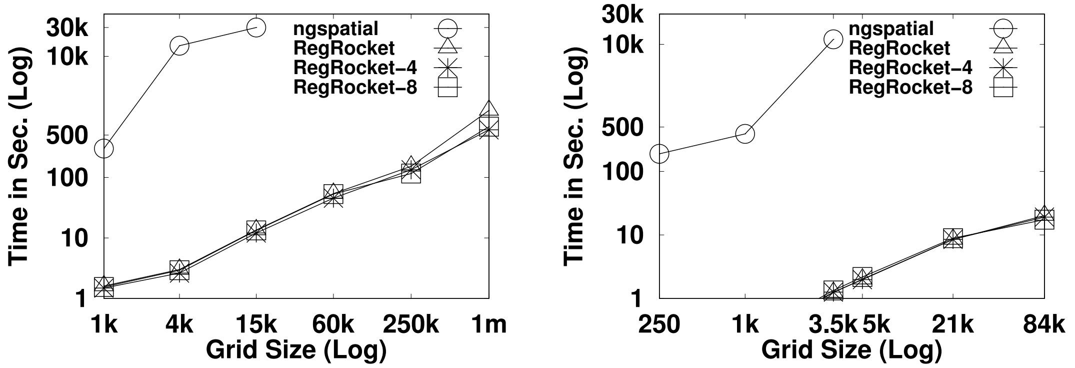
In this section, we compare the performance, both accuracy and scalability, of basic *RegRocket* and two generalized variations *RegRocket-8* and *RegRocket-4* with ngspatial, while scaling up the prediction grid size. In each experiment, either accuracy or scalability, we report the average of 5 different runs (we follow the same approach in all the experiments in our work).

Tables 5.5 and 5.6 show the precision, recall and F1-score values for each algorithm while scaling the grid size from 1k to 1 million cells in case of MNLandCover dataset, and from 250 to 84k cells in case of Ebird one, respectively. In all grid sizes, *RegRocket* and its variants *RegRocket-8* and *RegRocket-4* were able to significantly achieve better precision, recall and F1-score results than ngspatial. Specifically, in both datasets, *RegRocket* variants have an average precision of 0.87, recall of 0.92, and F1-score of 0.85, while ngspatial has an average precision of 0.56, recall of 0.79, and F1-score of 0.62. This indicates the efficiency of *RegRocket* in representing multinomial autologistic regression models. Note that the ngspatial results are incomplete after a grid size of 15k cells in case of the MNLandCover dataset, and 3.5k cells in case of the Ebird one, because of a failure in satisfying the memory requirements needed for its internal computations. We can also observe that the F1-score achieved by any *RegRocket* variation in both datasets is at least 0.65, which happens in the MNLandCover dataset with 1k cells, and can reach up to 0.95 at some cases. In general, the accuracy for small datasets tend to be lower than large ones due to the small number of grid cells used to train the model. As we can see from the table, the basic *RegRocket* has at maximum 20% lower F1-score than both *RegRocket-4* and *RegRocket-8*. The reason for that is the basic *RegRocket* captures less accurate neighbourhood dependencies than both of them. Note that *RegRocket-4* and *RegRocket-8* have very close accuracy results in some cases. This happens when the significant information between neighbourhoods with degrees 8 and 4 is very little, which makes the accuracy in the two cases are pretty similar.

Figures 5.4(a) and 5.4(b) depict the running time performance of each algorithm while using the same grid sizes in tables 5.5 and 5.6. We can observe that the three *RegRocket* variants and ngspatial have an average running time of 14 minutes and 8 hours, respectively. This means that *RegRocket* is at least 34 times faster than ngspatial. The poor performance of ngspatial comes

Grid Size	Metric	ngspatial	*RegRocket*	*RegRocket-4*	*RegRocket-8*
$1k$	Prec.	0.498	0.746	0.872	**0.731**
	Rec.	0.491	0.757	0.837	**0.763**
	F1	0.476	0.653	0.708	**0.683**
$4k$	Prec.	0.667	0.803	0.808	**0.933**
	Rec.	0.601	0.834	0.856	**0.871**
	F1	0.606	0.742	0.704	**0.782**
$15k$	Prec.	0.671	0.804	0.906	**0.962**
	Rec.	0.741	0.832	0.898	**0.903**
	F1	0.635	0.721	0.841	**0.834**
$60k$	Prec.	N/A	0.822	0.913	**0.976**
	Rec.	N/A	0.821	0.919	**0.919**
	F1	N/A	0.678	0.736	**0.798**
$250k$	Prec.	N/A	0.864	0.932	**0.967**
	Rec.	N/A	0.893	0.912	**0.915**
	F1	N/A	0.839	0.781	**0.806**
$1m$	Prec.	N/A	0.878	0.929	**0.961**
	Rec.	N/A	0.908	0.931	**0.895**
	F1	N/A	0.859	0.868	**0.873**

Table 5.5: Effect of Grid Size on Accuracy for the MNLandCover dataset.



(a) MNLandCover Dataset

(b) Ebird Dataset

Figure 5.4: Effect of Grid Size on Scalability.

from two reasons: (1) although ngspatial relies on parallel processing in its sampling, prior estimation and parameters optimization steps, it runs a centralized approximate Bayesian inference algorithm [10]. In contrast, *RegRocket* is a fully distributed framework. (2) ngspatial requires

Grid Size	Metric	ngspatial	*RegRocket*	*RegRocket-4*	*RegRocket-8*
250	Prec.	0.551	0.846	0.847	**0.858**
	Rec.	0.951	0.966	0.976	**0.985**
	F1	0.698	0.902	0.907	**0.917**
$1k$	Prec.	0.503	0.801	0.876	**0.883**
	Rec.	0.981	0.986	0.965	**0.961**
	F1	0.665	0.884	0.918	**0.921**
$3.5k$	Prec.	0.477	0.865	0.916	**0.901**
	Rec.	0.977	0.991	0.992	**0.985**
	F1	0.641	0.924	0.952	**0.941**
$5k$	Prec.	N/A	0.885	0.875	**0.912**
	Rec.	N/A	0.984	0.986	**0.984**
	F1	N/A	0.932	0.927	**0.947**
$21k$	Prec.	N/A	0.864	0.866	**0.895**
	Rec.	N/A	0.984	0.991	**0.991**
	F1	N/A	0.921	0.924	**0.941**
$84k$	Prec.	N/A	0.889	0.929	**0.919**
	Rec.	N/A	0.991	0.993	**0.991**
	F1	N/A	0.937	0.956	**0.954**

Table 5.6: Effect of Grid Size on Accuracy for the Ebird dataset.

estimating a prior distribution for each predictor variable, and hence it suffers from a huge latency before starting the actual learning process. Note that the ngspatial curve is incomplete for grids with sizes more than 15k cells in case of the MNLandCover dataset, and 3.5k cells in case of the Ebird one as in tables 5.5 and 5.6. We also find that in case of datasets with large grids (e.g., 1 million cells in the MNLandCover dataset), both *RegRocket-4* and *RegRocket-8* achieve lower latency overhead than basic *RegRocket*. For example, at 1 million case, *RegRocket-4* and *RegRocket-8* variations are two times faster on average. This is because increasing the neighbourhood degree leads to producing less number of predicates (See tables 5.3 and 5.4), and hence less number of factor graph nodes to process, which makes the weights learning process faster. In this experiment, the performance of the three *RegRocket* variations are almost similar in case of small grid sizes (i.e., the average accuracy difference between the three variations is less than 20 seconds). However, the difference becomes significant in the case of large grid sizes (average of 600 seconds difference for grid size of 1 million cells). This shows that *RegRocket* is efficient when scaling up the grid size regardless of the neighbourhood degree. Note that both figures 5.4(a) and 5.4(b) follow a log-scale.

Num. of Epochs	Metric	*RegRocket*	*RegRocket-4*	*RegRocket-8*
100	Prec.	0.815	0.883	0.906
	Rec.	0.845	0.864	0.854
	F1	0.772	0.732	0.715
1000	Prec.	**0.864**	**0.932**	**0.967**
	Rec.	**0.893**	**0.912**	**0.915**
	F1	**0.839**	**0.781**	**0.806**
$10k$	Prec.	0.881	0.931	0.966
	Rec.	0.866	0.909	0.915
	F1	0.826	0.785	0.795

Table 5.7: Effect of Learning Epochs on Accuracy for the MNLandCover dataset.

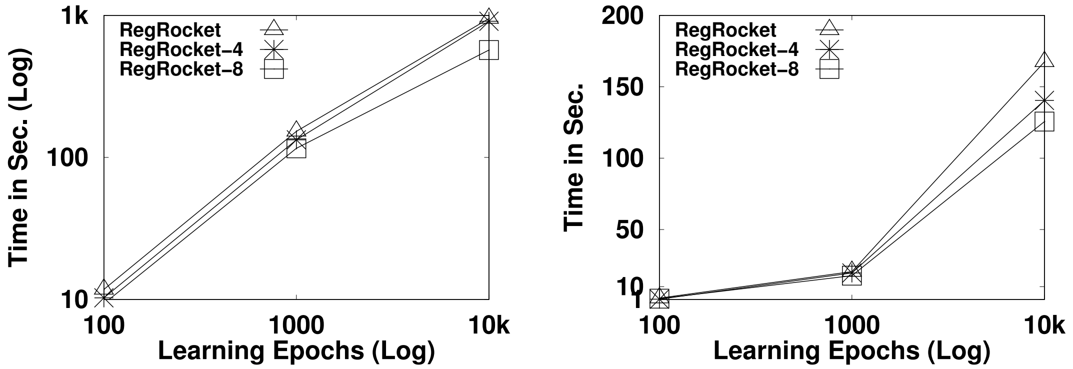
Num. of Epochs	Metric	*RegRocket*	*RegRocket-4*	*RegRocket-8*
100	Prec.	0.849	0.899	0.909
	Rec.	0.845	0.835	0.825
	F1	0.847	0.866	0.865
1000	Prec.	**0.889**	**0.929**	**0.919**
	Rec.	**0.991**	**0.993**	**0.991**
	F1	**0.937**	**0.961**	**0.954**
$10k$	Prec.	0.909	0.919	0.919
	Rec.	0.925	0.935	0.995
	F1	0.917	0.927	0.955

Table 5.8: Effect of Learning Epochs on Accuracy for the Ebird dataset.

Effect of Learning Epochs E

In this section, we evaluate the performance, both accuracy and scalability, of basic *RegRocket*, *RegRocket-4* and *RegRocket-8*, while having three different values of learning epochs. In the following experiments, we fix the grid size in both datasets to the default values.

Tables 5.7 and 5.8 show the values of accuracy metrics for the three variations of *RegRocket* while changing the number of epochs from 100 to 10k. The results show an interesting observation that all variations of *RegRocket* can rapidly converge to their optimal values of weights (i.e., number of learning epochs 1000 only). The rapid convergence happens because weights are shared among all locations which makes their values updated multiple times using the gradient descent optimization in each epoch. As a result, *RegRocket* just needs a small number of epochs for weights convergence. In general, basic *RegRocket* needed a higher number of



(a) MNLandCover Dataset

(b) Ebird Dataset

Figure 5.5: Effect of Learning Epochs on Scalability.

epochs (i.e., 10k), compared to *RegRocket-4* and *RegRocket-8*, to achieve higher accuracy. This matches our performance hint that generalized variations such as *RegRocket-4* and *RegRocket-8* could be more efficient than the basic *RegRocket*.

Figures 5.5(a) and 5.5(b) show the running time for the different variations given the same setup in tables 5.7 and 5.8. In general, *RegRocket* with all variations is extremely efficient because of the parallel processing of learning epochs in *RegRocket*. However, we observe that *RegRocket-8* significantly outperforms *RegRocket-4* and *RegRocket*. It is at least 40% and 25% faster than both of them in the MNLandCover and Ebird datasets, respectively. Note that Figure 5.5(a) follows a log-scale, but Figure 5.5(b) is not.

Effect of Optimization Step Size α

In this section, we evaluate the performance, both accuracy and scalability, of the different variations of *RegRocket* while varying the value of the step size α that is used during the execution of gradient decent optimization. We use the same datasets used in the previous experiments.

Tables 5.9 and 5.10 depict the effect of increasing the value of step size from 0.0001 to 0.1. In general, the accuracy of all *RegRocket* variations decreases by increasing the step size in both datasets. We also observe that the highest prediction accuracy tends to saturate in most cases at value 0.001. The main reason behind this is that large step sizes lead to large updates while optimizing the weights and hence they cannot smoothly converge to the optimal values.

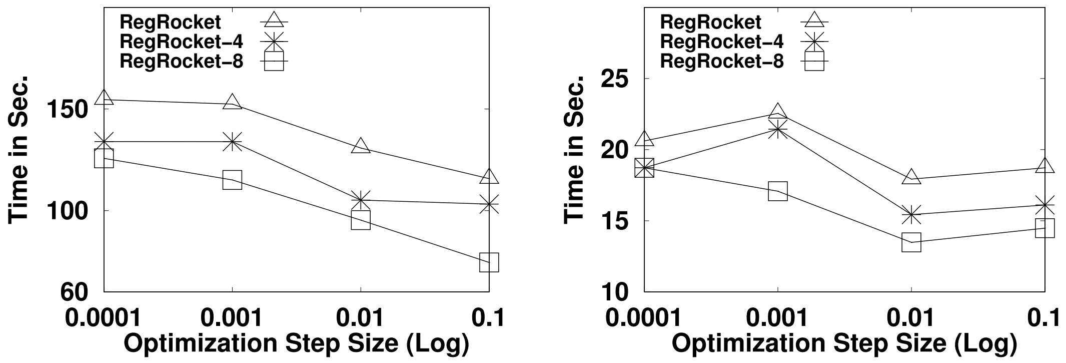
Step Size	Metric	*RegRocket*	*RegRocket-4*	*RegRocket-8*
0.0001	Prec.	0.829	0.921	0.966
	Rec.	0.816	0.789	0.915
	F1	0.782	0.825	0.875
0.001	Prec.	**0.864**	**0.932**	**0.967**
	Rec.	**0.893**	**0.912**	**0.915**
	F1	**0.839**	**0.781**	**0.806**
0.01	Prec.	0.819	0.871	0.926
	Rec.	0.806	0.838	0.875
	F1	0.756	0.745	0.795
0.1	Prec.	0.779	0.861	0.916
	Rec.	0.766	0.828	0.865
	F1	0.676	0.745	0.785

Table 5.9: Effect of Optimization Step Size on Accuracy for the MNLandCover dataset.

Step Size	Metric	*RegRocket*	*RegRocket-4*	*RegRocket-8*
0.0001	Prec.	0.914	0.909	0.929
	Rec.	0.993	0.998	0.995
	F1	0.952	0.951	0.961
0.001	Prec.	**0.889**	**0.929**	**0.919**
	Rec.	**0.991**	**0.993**	**0.991**
	F1	**0.937**	**0.956**	**0.954**
0.01	Prec.	0.879	0.909	0.899
	Rec.	0.985	0.985	0.985
	F1	0.929	0.945	0.941
0.1	Prec.	0.779	0.884	0.879
	Rec.	0.985	0.895	0.895
	F1	0.871	0.889	0.887

Table 5.10: Effect of Optimization Step Size on Accuracy for the Ebird dataset.

We conclude that the step size should be kept relatively small on average in *RegRocket*. However, this comes with higher latency as in figures 5.6(a) and 5.6(b) that show the corresponding running times. For example, decreasing the step size from 0.01 to 0.001 in the MNLandCover dataset incurs 26% additional latency overhead while running *RegRocket-4*.



(a) MNLandCover Dataset

(b) Ebird Dataset

Figure 5.6: Effect of Optimization Step Size on Scalability.

Num. of Partitions	Metric	*RegRocket*	*RegRocket-4*	*RegRocket-8*
50	Prec.	0.945	0.962	0.971
	Rec.	0.894	0.931	0.912
	F1	0.852	0.877	0.914
100	Prec.	0.913	0.954	0.961
	Rec.	0.891	0.923	0.931
	F1	0.843	0.812	0.861
200	Prec.	**0.864**	**0.932**	**0.967**
	Rec.	**0.893**	**0.912**	**0.915**
	F1	**0.839**	**0.781**	**0.806**
300	Prec.	0.782	0.812	0.815
	Rec.	0.734	0.831	0.821
	F1	0.689	0.701	0.712

Table 5.11: Effect of Number of Factor Graph Partitions on Accuracy for the MNLandCover dataset.

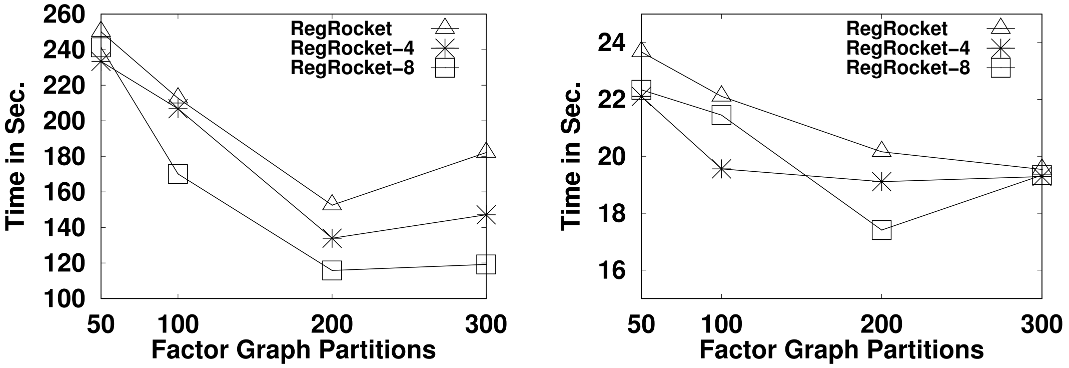
Effect of Number of Factor Graph Partitions C

In this section, we evaluate the performance, both accuracy and scalability, of the different variations of *RegRocket* while varying the number of factor graph partitions C from 50 to 300. We use the same datasets used in the previous experiments with the default configurations.

Tables 5.11 and 5.12 show the effect of increasing the number of factor graph partitions on the precision, recall and F1-score values. We observe that the F1-score values decrease when increasing the number of partitions because the number of predicates that are replicated

Num. of Partitions	Metric	*RegRocket*	*RegRocket-4*	*RegRocket-8*
50	Prec.	0.967	0.944	0.968
	Rec.	0.992	0.991	0.982
	F1	0.979	0.967	0.975
100	Prec.	0.923	0.941	0.937
	Rec.	0.971	0.981	0.983
	F1	0.946	0.961	0.959
200	Prec.	**0.889**	**0.929**	**0.919**
	Rec.	**0.991**	**0.993**	**0.991**
	F1	**0.937**	**0.959**	**0.953**
300	Prec.	0.674	0.789	0.792
	Rec.	0.782	0.712	0.812
	F1	0.724	0.748	0.802

Table 5.12: Effect of Number of Factor Graph Partitions on Accuracy for the Ebird dataset.



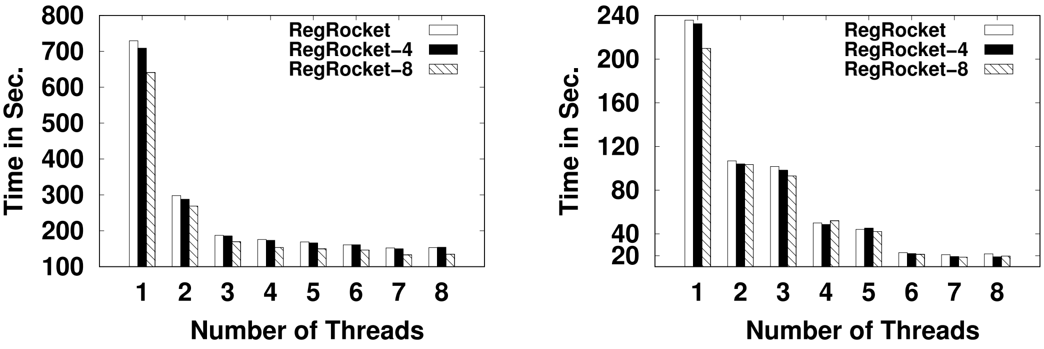
(a) MNLandCover Dataset

(b) Ebird Dataset

Figure 5.7: Effect of Number of Factor Graph Partitions on Scalability.

among partitions increases. This results in more iterations to update the weights as in shown in Algorithm 5 (Lines 15 to 25), which makes the weights suffer from an over-fitting issue, and hence the accuracy decreases. For example, in both datasets, when increasing the number of partitions from 200 to 300, the F1-scores in case of *RegRocket*, *RegRocket-8* and *RegRocket-4* decrease by 17%, 10% and 11%, respectively, on average.

Figures 5.7(a) and 5.7(b) depict the running time for the different variations of *RegRocket* given the same setup in tables 5.11 and 5.12. Increasing the number of partitions leads to more parallelism, and hence the running time starts to decrease. However, increasing the number of



(a) MNLandCover Dataset

(b) Ebird Dataset

Figure 5.8: Effect of Number of Threads on Scalability.

partitions after a certain threshold (e.g., 200) makes the running time overhead to handle the predicates replication significant, and hence the whole running time slightly increases again. For example, in both datasets, the average F1-score decrease is 30% in all variations when changing the number of partitions from 100 to 200. After that, the F1-score starts to increase due to the replication overhead. In our experiments, we choose the number of partitions to be 200 in order to balance between the accuracy and the running time.

Effect of Number of Threads

Figures 5.8(a) and 5.8(b) show the effect of increasing the number of threads from 1 to 8 on the three variations of *RegRocket* for both datasets. These threads are used to parallelize the work in the weights learner module of *RegRocket*. As expected, the performance of all variations significantly improves by increasing the number of threads. For example, the running time of the basic *RegRocket* using 8 threads is at least 4 times faster than using 1 thread in the MNLandCover dataset. This shows the ability of *RegRocket* to scale up with system threads. Note that the performance difference between 7 and 8 threads is almost the same because all cores are exploited in both cases.

5.6 Conclusions

This chapter presented *RegRocket*, a scalable framework for building multinomial autologistic regression models to predict spatial categorical data. *RegRocket* focuses on the autologistic models that consist of prediction and predictor variables with unordered categories. *RegRocket* provides an efficient modeling for the multinomial autologistic regression problem using Markov Logic Networks (MLN), which is a scalable statistical learning framework. *RegRocket* employs an efficient gradient descent-based optimization technique to learn the autologistic model parameters. Experimental analysis using real data sets shows that *RegRocket* is scalable and provides accurate capturing for the multinomial autologistic regression problem.

Algorithm 5 Function UPDATEWEIGHTS (Variable v, VariablesIndex $\mathcal{V}$, PredicatesIndex $\mathcal{P}$, StepSize α)

```
if $v$ is not a prediction variable for a pivot outcome, and belongs to the training data then
    $l_i \leftarrow \mathcal{V}[v]$.location, $g \leftarrow 1$ /* Gradient Value */
    $\hat{z}_i(\lambda_v) \leftarrow$ Prediction of outcome $\lambda_v$ at $l_i$ using $\beta$ and $\eta$ (Equation 5.1)
    if $\mathcal{V}[v]$.value $\neq \hat{z}_i(\lambda_v)$ then
        $g \leftarrow$ -1
    end if
    if $v$ is any predictor variable $x_j^{\lambda_v,t_v}(i) \in \mathcal{X}(i)$ then
        $\beta_j^{\lambda_v,t_v} \leftarrow \beta_j^{\lambda_v,t_v}$ + $\alpha$ $g$ /* Gradient Descent on $\beta_j^{\lambda_v,t_v}$*/
    else
        for each $\beta_j^{\lambda_v,t_v} \in \beta$ do
            $\beta_j^{\lambda_v,t_v} \leftarrow \beta_j^{\lambda_v,t_v}$ + $\alpha$ $g$ /* Gradient Descent on $\beta_j^{\lambda_v,t_v}$*/
        end for
    end if
    if $v$ is prediction variable $z_i(\lambda_v)$ then
        for each $p \in \mathcal{P}[v]$ do
            if $p$ is a neighbour-based predicate then
                $\hat{z_k}(s_p) \leftarrow$ Prediction of outcome $s_p$ at neighbour $l_k$ in $p$ using $\beta$ and $\eta$
                if $\mathcal{V}[v_k]$.value $\neq \hat{z_k}(s_p)$ then
                    $g \leftarrow$ -1
                else
                    $g \leftarrow 1$
                end if
                $\eta_{\lambda_v,s_p} \leftarrow \eta_{\lambda_v,s_p}$ + $\alpha$ $g$ /* Gradient Descent on $\eta_{\lambda_v,s_p}$ */
            end if
        end for
    end if
end if
```

Chapter 6

Flash: Scalable Spatial Data Analysis Using Markov Logic Networks

6.1 Introduction

Spatial data analysis has grabbed significant attention from both industry and academia (see [180] for a comprehensive survey). The main objective is to extract insights and useful patterns from spatial data (e.g., satellite images [15], geotagged tweets [5]). Spatial data analysis has been employed in many crucial applications in different domains (e.g., public health [142, 181, 182], transportation [183, 184], environmental science [185–187], climatology [188–190], market analytics [191], and biology [192]). For example, environmentalists analyze geotagged tweets to predict the people who might need help during disasters [193]. Epidemiologists use spatial analysis techniques to identify cancer clusters [23]. As a result, researchers and practitioners worldwide have released many spatial analysis systems and libraries. Examples of these systems include GeoDa [94], spBayes [194], GDAL [195], PySAL [196], ngspatial [51], CrimeStat [197], ESRI ArcGIS [198] (see [180, 199] for a comprehensive survey).

Existing spatial analysis solutions suffer from two main issues. First, they can not scale beyond implementing prototypes over small spatial datasets (e.g., see [51, 55]) (*scalability issue*). The scalability challenge is mainly because these solutions were not originally designed for the huge amounts of spatial data being generated at the moment (e.g., there are 10 Million geotagged tweets issued every day [5]). Second, these solutions are specifically tailored for domain-specific applications (e.g., a spatial hidden Markov model for animal tracking [200], and a statistical learning approach for crime analysis [197]) (*non-generic issue*). As a result, to employ a spatial analysis technique in a new application, a developer/user would need to re-implement and optimize such technique at the application layer. This is inconvenient for a non-expert application developer who might not be quite familiar with efficient implementations of spatial data analysis techniques.

In this chapter, we present *Flash*; a framework for *generic* and *scalable* spatial data analysis. *Flash* achieves orders of magnitudes scalability gain over existing solutions while preserving the same accuracy. It focuses on building a major class of spatial analysis techniques, called *spatial probabilistic graphical modeling* (SPGM), which uses probability distributions and graphical representations (e.g., spatial Bayesian networks [201]) to describe spatial phenomena and make predictions about them [180]. SPGM has many applications including health care [12], risk analysis [57], and environmental science [58].

Flash exploits the Markov Logic Networks (MLN) framework [1] to express SPGM with logical semantics, and allow developers to implement their applications using a set of rules instead of thousands of lines of code. To support *scalability*, *Flash* translates the generated MLN rules of any SPGM application into SQL queries using a grounding technique from [36], and then executes these queries inside scalable database engines (e.g., PostgreSQL). In addition, *Flash* provides spatial variations of the RDBMS-based learning and inference algorithms of MLN [66] to perform scalable SPGM predictions (e.g., predictions over probabilistic models with millions of nodes). Using *Flash*, a myriad of spatial applications can be built without the need to worry about the underlying SPGM computation. To show the effectiveness of *Flash*, we use it to provide the MLN-based representation of three popular SPGM models including spatial Markov random fields (SMRF) [11], spatial hidden Markov models (SHMM) [12] and spatial Bayesian networks (SBN) [55].

We experimentally evaluate *Flash* by building three spatial analysis applications, where *Flash* is used to implement their underlying SPGM: (1) *Bird monitoring*: an application that uses spatial Markov random fields (SMRF) to predict the existence of a specific bird species, namely Barn Swallow, over North America. This application uses Ebird dataset [150] containing more than 360 Million bird observations at 84K location cells. (2) *Safety analysis*: an application that uses spatial hidden Markov models (SHMM) to determine the safety level at different locations based on the reported incidents. As a case study, we assess the safety in Chicago based on its official crime data repository [202], that has around 7 Million reported incidents. (3) *Land use change tracking*: an application that uses spatial Bayesian networks (SBN) to analyze where the change in land use is most likely to occur. This application uses a grid dataset containing one Million cells of land cover distribution over Minnesota state, and is compiled from national land cover data repository [171]. For each application, we compare the accuracy and scalability of the built SPGM models using *Flash* and the state-of-the-art computational methods, where we show the efficiency of *Flash* in building SPGM models.

The rest of this chapter is organized as follows: Section 6.2 gives an overview of the *Flash* architecture. Section 6.3 describes the MLN-based implementations of SMRF, SHMM and SBN models as case studies for *Flash*. Section 6.4 provides an experimental evaluation of *Flash*. Finally, Section 6.5 concludes the chapter.

6.2 Flash Overview

Figure 6.1 depicts the system architecture of *Flash*. It has two types of users; *administrator* and *client*. An *administrator* should have expertise with both MLN and SPGM to provide user-defined functions for transforming spatial graphical models into a set of first-order logical rules [14]. A *client* can be either application developer or casual user. She can build the SPGM of any application by specifying some settings as input. The built model will be stored in a relational database (e.g., PostgreSQL) as a factor graph [7]. A client can also issue learning and prediction queries over the built models. Learning queries can fit the parameters of a specific model to input application data (e.g., hidden Markov model parameters). Prediction queries can answer relevant questions about the model (e.g., what is the probability of a specific event to happen?). As depicted in Figure 6.1, *Flash* consists of the following four main modules:

MLN Transformation. For any SPGM input, this module is responsible on generating an equivalent set of *weighted* rules containing logical predicates (e.g., bitwise-AND, and imply). These weights represent the original SPGM parameters. The generated rules follow the syntax of an efficient Datalog-like logic programming framework, called DDlog [2]. *Flash* chooses DDlog because of its DBMS-friendly schema declaration and rules syntax that can be efficiently processed during the model building module. Currently, *Flash* supports transformation for three spatial graphical models; spatial Markov random fields (SMRF) [11], spatial hidden Markov models (SHMM) [12], and spatial Bayesian networks (SBN) [55] (Details are in Section 6.3).

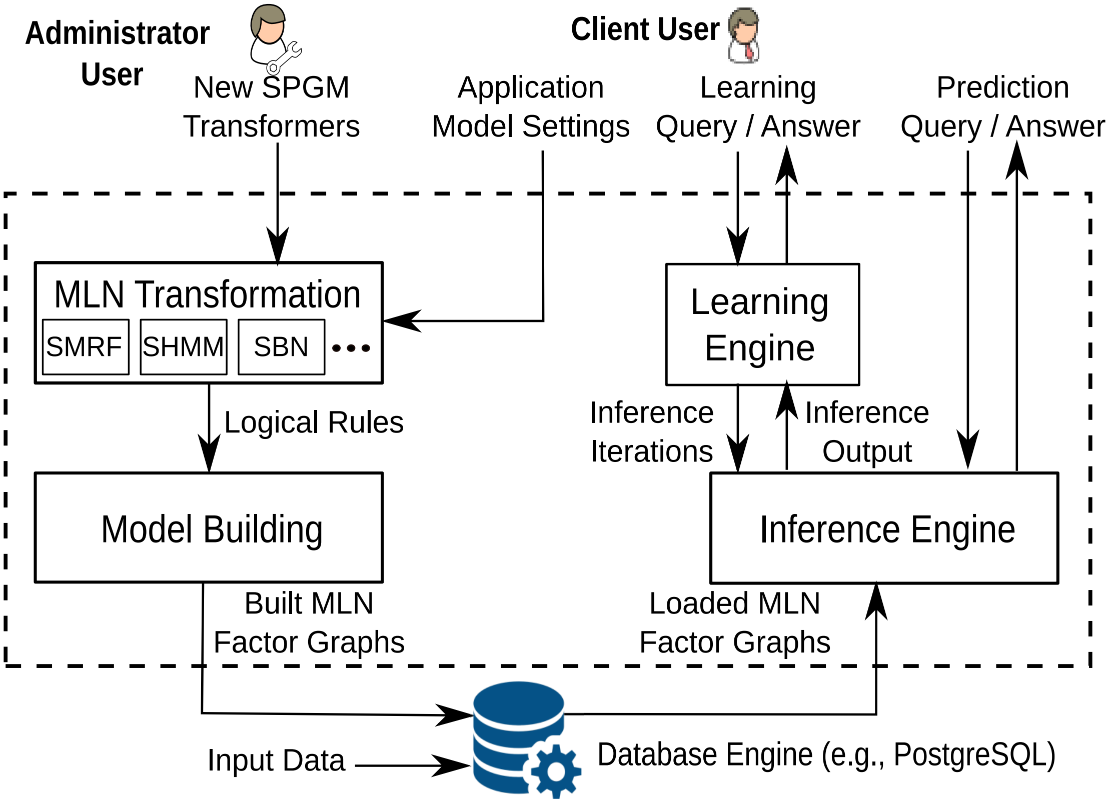
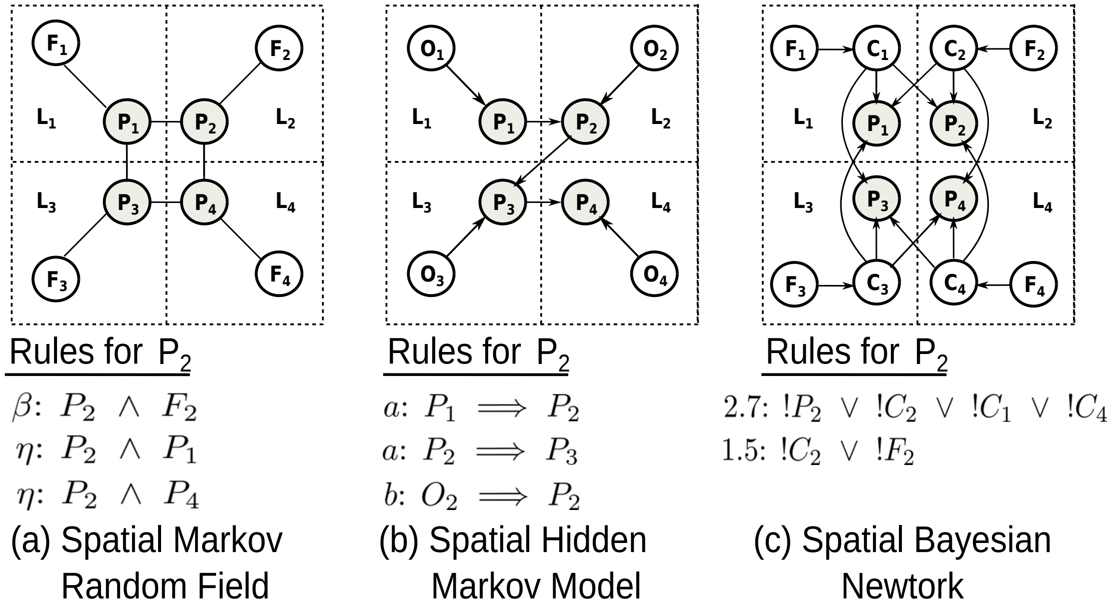


Figure 6.1: *Flash* System Overview.

Model Building. The generated logical rules from the MLN transformation module are considered templates for constructing factor graphs [7]. As a result, *Flash* adapts a scalable factor graph grounding technique from [36] to efficiently translate these rules into SQL queries, and then apply such queries on the input application data to obtain the final output that is equivalent to the SPGM input.

Inference Engine. *Flash* evaluates the prediction queries using Gibbs sampling-based inference algorithms over factor graphs [66]. However, such algorithms perform sequential sampling over the factor graph nodes which results in slow convergence to the inference answer in case these nodes have spatial dependencies as in SPGM applications [88]. To overcome this limitation, *Flash* employs a variation of Gibbs Sampling that exploits a *concliques*-based traversal pattern [88] to efficiently sample spatially-dependent nodes. A conclique is a set of nodes such that no two nodes in this set are spatially neighbours. The main idea behind defining concliques



Rules for P_2

β: $P_2 \wedge F_2$

η: $P_2 \wedge P_1$

η: $P_2 \wedge P_4$

(a) Spatial Markov Random Field

Rules for P_2

a: $P_1 \Longrightarrow P_2$

a: $P_2 \Longrightarrow P_3$

b: $O_2 \Longrightarrow P_2$

(b) Spatial Hidden Markov Model

Rules for P_2

2.7: $!P_2 \vee !C_2 \vee !C_1 \vee !C_4$

1.5: $!C_2 \vee !F_2$

(c) Spatial Bayesian Newtork

Figure 6.2: SMRF, SHMM, and SBN Representations in *Flash*.

is ensuring the neighbouring independence between nodes in the same conclique set, and hence these nodes can be sampled in parallel.

Learning Engine. *Flash* employs a pseudo-likelihood learning algorithm to learn any unknown weights of the generated MLN rules (i.e., SPGM parameters) from the factor graph. This algorithm repeatedly uses the proposed spatial variation of Gibbs sampling-based inference algorithm in the inference engine to compute the gradient of the SPGM pseudo-likelihood and then determine the weights using an efficient gradient descent optimization technique.

6.3 Case Studies in Flash

Flash supports the implementation of three common spatial graphical models; spatial Markov random fields (SMRF) [11], spatial hidden Markov models (SHMM) [12], and spatial Bayesian networks (SBN) [55], as case studies. Figure 6.2 gives toy examples on the logical representation of these three models in *Flash*, where each model is defined over 4-cells grid, and the neighborhood of any cell l is assumed to be the cells that share edges with l only.

6.3.1 Spatial Markov Random Field (SMRF)

Spatial Markov random fields (SMRFs) are powerful and important tools for modeling spatial data and building analysis applications. They have been widely used in different areas of spatial statistics [161, 165, 167]. As with many other areas of statistics, a major challenge for spatial analysts is dealing with massive data sets. This is particularly problematic for SMRFs due to the need for matrix operations that involve very large matrices can be computationally prohibitive, specially in the case of Gaussian processes. Existing approaches tried to solve the scalability issues in two categories. The first category focused on developing fast matrix computations that exploit the sparsity of matrices in SMRF models (e.g., [203, 204]). However, utilizing sparsity does not seem to be among the more promising strategies as it does not fit the dense data cases. The second category introduced fast likelihood approximations for the Gaussian-based SMRF models (e.g., [205, 206]). However, this category is not generic enough to capture other arbitrary SMRF models (i.e., the interactions between random variables in the SMRF model are not captured with multivariate Gaussian distributions).

In contrast to these existing approaches, *Flash* provides a scalable approach for SMRF models by introducing a first-order logic representation, where there is an equivalent weighted *bitwise-AND* predicate for each pair of connected variables. In this case, the predicates' weights correspond to the SMRF parameters that need to be learned. Figure 6.2(a) shows a small SMRF model with a prediction P_l and feature F_l at each cell l. Each prediction P_l has undirected edges with feature F_l at this cell and each neighboring prediction variable. For example, P_2 is connected with feature F_2 and neighbors P_1 and P_4.

6.3.2 Spatial Hidden Markov Models (SHMM)

The hidden Markov model (HMM) is a doubly embedded stochastic method based on probability theory, which can be used in a sequence labeling problem. It describes the process of randomly generating non-observable state sequences from a hidden Markov chain and generating an observation from each state to produce an observable sequence. In spatial hidden Markov model (SHMM), the sequences are generated on spatially-correlated random variables.

While all existing SHMM solutions are innovative, they face severe scalability issues when dealing with big spatial data. The scalability challenge is mainly because these solutions were not originally designed for the big data era or to exploit new high performance computing environments [207]. In contrast, *Flash* scales up the performance of SHMM by providing an equivalent MLN representation, where any state/state or observation/state pair is mapped to a weighted *imply* predicate, and the resulting weights correspond to the SHMM parameters. Figure 6.2(b) shows a small SHMM model with a hidden state P_l and observation O_l variables at each cell l. Each observation O_l has a directed edge to state P_l at this cell. In addition, SHMM imposes an ordered spatial dependence among neighboring locations, where it uses z-curve ordering technique to build a sequence that preserves the spatial dependence between prediction variables (e.g., P_1 has a directed edge to P_2, and P_2 has another one to P_3, etc).

6.3.3 Spatial Bayesian Networks (SBN)

Numerous applications model the probability of an input event to occur based on other *causal* events that have spatial dependencies with the input event. These applications include meteorology [56], risk analysis [57, 208], and environmental science [58, 209–211]. For example, wildlife managers determine the success probability of a certain conservation plan based on events about the distribution of animal species in neighboring locations [211]. Business analysts forecast the budget and likely costs of water infrastructure networks based on failure events in water mains at neighboring sites [208]. A typical solution to model the *causal dependencies* between events in all these applications is to employ spatial Bayesian networks (a.k.a spatial Bayesian belief networks) [13, 212]. These networks are directed probabilistic graphs whose nodes represent variables corresponding to events over neighboring locations, and the edges represent the casual relationships between these variables. For example, two events ”rain” and ”flood” at neighboring locations x and y, respectively, can be represented as two random variables, where the *rain* variable is a cause (i.e., parent node in the graph) for the *flood* variable.

Existing solutions of spatial Bayesian networks can not scale beyond implementing prototypes over small spatial and spatio-temporal datasets [13, 55]. Meanwhile, Markov Logic Networks (MLN) are recently used to scale up the performance of classical Bayesian networks that do not consider spatial dependencies between random variables (e.g., Bayesian Logic Networks [213] and ProbCog [214]).

In *Flash*, we exploit Markov Logic Networks (MLN) to represent the SBN models. *Flash* provides an equivalent weighted combination of *bitwise-OR* and *negation* predicates for each causality relation (i.e., directed edge). The weights of these predicates are calculated from the input prior probabilities of SBN. Figure 6.2(c) shows a small SBN model with a prediction variable P_l at each cell l which is affected directly by a status variable C_l and indirectly by a feature variable F_l (i.e., F_l has a direct edge to C_l). In addition, each prediction P_l is affected by the status variables at the neighboring cells.

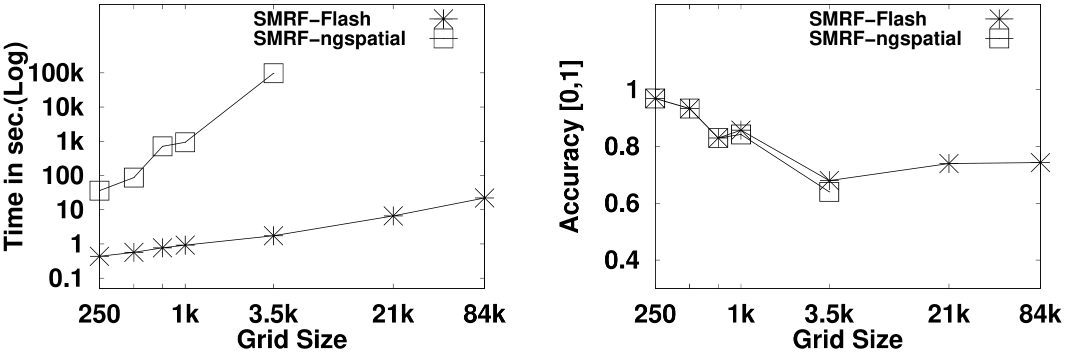
6.4 Experiments

In this section, we experimentally evaluate the accuracy and scalability of *Flash* in building SPGM models for three spatial analysis applications. In these applications, we compare the performance of *Flash* with ngspatial [51], shmm [200], and bnspatial [55] tools when building SMRF [11], SHMM [12], and SBN [55] models, respectively.

6.4.1 Experimental Setup

Applications. The details of the three applications, along with their datasets, used in our experiments are described as follows:

- *Bird Monitoring.* This application predicts the existence of a bird species across a certain area. Ornithologists model this problem using SMRF [51] as shown in [215], where the area is divided by a two-dimensional grid. Each grid cell holds a binary prediction variable indicating the presence or absence of the bird at this cell, and a set of feature variables that help predicting the value of this prediction variable. Then, the prediction at any cell is determined based on the values of feature variables at this cell along with a set of predicted or observed values at neighbouring cells. As a case study, we use the daily distribution of a certain bird species, namely Barn Swallow, from Ebird dataset [150], which contains more than 360 Million observations collected over North America. We define a grid of 84K cells, and map each observation to one cell. Then, we build the SMRF-based prediction model of the bird existence at cells with no observations.
- *Safety Analysis.* The objective of this application is to infer the safety level (e.g., low, medium and high) at a bunch of neighbouring locations simultaneously based on reported incidents at these locations. This application has been usually represented with SHMM [12] as shown in [216], where the safety level at each location is considered a hidden state to be predicted and the reported incident at this location is an observation that affects the prediction value. As a case study, we use the official Chicago crime dataset



(a) Grid Size vs. Scalability

(b) Grid Size vs. Accuracy

Figure 6.3: Effect of Grid Size on Scalability and Accuracy for SMRF Model.

repository [202], which contains around 7 Million reported incidents (i.e., observations) over 500K grid locations.

- *Land Use Change Tracking.* The objective of this application is to determine whether there will be a change in the land use or not. For example, the land in a location l could be suitable for agriculture, however, given certain factors (e.g., crowded neighbourhoods), it is expected to be for human use soon. We model this application as SBN problem. As a case study, we use a grid dataset containing one Million cells of land cover distribution over Minnesota state, and is compiled from national land cover data repository [171].

In each application, we randomly select 15% of the grid cells of its input data as testing data, and use the rest 85% for training any SPGM model.

Environment. 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.

Metrics. In all experiments, we use the total running time of learning the parameters of any SPGM model as a scalability evaluation metric, and the ratio of correctly predicted cells using the learned model to the total number of test cells as an accuracy evaluation metric.

6.4.2 Experimental Results

Study of SMRF Scalability and Accuracy

In this section, we compare the performance, both scalability and accuracy, of *Flash* with ngspatial [51], when learning and using the SMRF models that are built for five different sizes of Ebird grid data (Bird Monitoring Application).

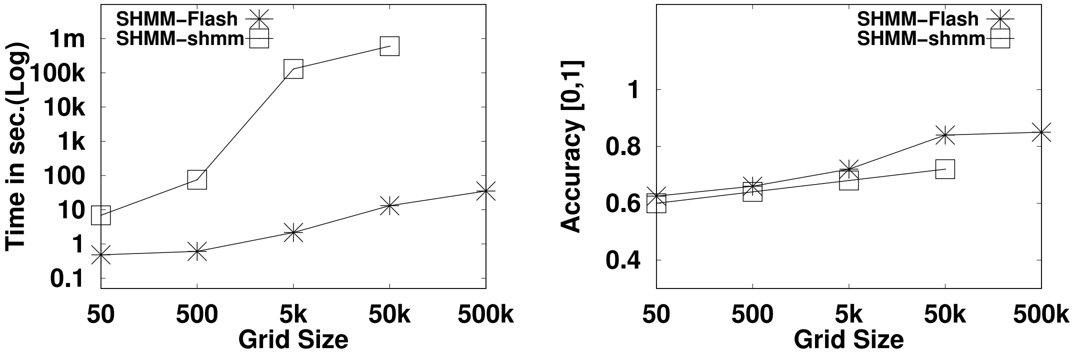
Figure 6.3(a) shows the running time for each algorithm to learn the SMRF parameters while scaling the grid size from 250 to 84k cells. For the five grid sizes, *Flash* was able to significantly reduce the running time compared to ngspatial. Specifically, *Flash* and ngspatial have an average running time of 4.7 seconds and 5.5 hours, respectively. This means that *Flash* has at least three orders of magnitude reduction in the running time over ngspatial. Note that the ngspatial curve in Figure 6.3(a) is incomplete after a grid size of $3.5k$ cells because of a failure in satisfying the memory requirements needed for its internal computations. In contrast, the running times for *Flash* are complete. This shows the *Flash* efficiency when scaling up the grid size regardless of the model specified.

Figure 6.3(b) shows the accuracy for each algorithm while using the same grid sizes in Figure 6.3(a). As mentioned before, we divide the cells in each grid into training and testing sets, where we randomly select 15% of cells for testing and keep the rest for training. We repeat this process 5 times and then average the accuracy results (we follow the same approach in the whole accuracy experiments in our work). As can be seen in the figure, *Flash* has almost the same accuracy achieved by ngspatial at small grid sizes, while it is slightly more accurate (4% more) than ngspatial at the grid size of $3.5k$ cells. Note that the ngspatial curve is incomplete for grids with sizes more than $3.5k$ cells as in Figure 6.3(a).

Study of SHMM Scalability and Accuracy

In this section, we compare the performance, both scalability and accuracy, of *Flash* with shmm [200], when learning and using the SHMM models that are built for five different sizes of Chicago crime grid data (Safety Analysis Application).

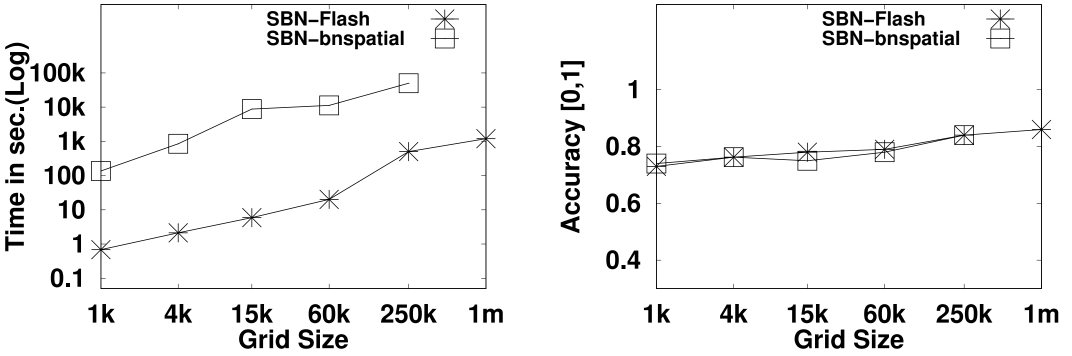
Figure 6.4(a) shows the running time for each algorithm while scaling the grid size from 50 to $500k$ cells. We can observe from the results that *Flash* has an average three orders of magnitude less running time than shmm. In contrast to ngspatial in Figure 6.3(a), shmm is more scalable to relatively large grid sizes (e.g., $50k$ cells), but, still can not complete the running for huge grid sizes like $500k$ cells.



(a) Grid Size vs. Scalability

(b) Grid Size vs. Accuracy

Figure 6.4: Effect of Grid Size on Scalability and Accuracy for SHMM Model.



(a) Grid Size vs. Scalability

(b) Grid Size vs. Accuracy

Figure 6.5: Effect of Grid Size on Scalability and Accuracy for SBN Model.

Figure 6.4(b) shows the prediction accuracy for each algorithm while using the same grid sizes in Figure 6.4(a). We observe that *Flash* is consistently more accurate than shmm at all grid sizes, however, *Flash* has larger improvement ratio when the grid size becomes larger (the improvement ratio can reach to 18%). Note that the shmm curve is also incomplete for the grid with $500k$ cells as in Figure 6.4(a).

Study of SBN Scalability and Accuracy

In this section, we compare the performance, both scalability and accuracy, of *Flash* with bnspatial [55], when learning and using the SBN models that are built for six different sizes of Minnesota land use data (Land Use Tracking Application).

Figure 6.5(a) shows the running time for each algorithm while scaling the grid size from $1k$ to 1 million cells. In general, *Flash* is much faster than bnspatial in all cases, however the ratio of improvement in case of SBN is less than its counterpart in SHMM. We observe from the results that *Flash* has at least two orders of magnitude less running times than bnspatial. The running times of *Flash* range from 0.6 sec (minimum value) to 20 hours (maximum value).

Figure 6.5(b) shows the accuracy for each algorithm while using the same grid sizes in Figure 6.5(a). As shown in the figure, *Flash* does not improve so much over the accuracy already obtained by bnspatial. In general, the main objective of *Flash* is to speed up the computation steps of SPGM models, while keeping the same accuracy obtained by the best state-of-the-art techniques or increasing it, if possible.

6.5 Conclusions

The current explosion in spatial data raises the need for efficient spatial analysis tools to extract useful information from such data. In this chapter, we present *Flash*; a framework for scalable spatial data analysis, with a special focus on spatial probabilistic graphical modeling (SPGM), based on Markov Logic Networks (MLN). The main idea is to express any SPGM application as a set of first-order logical rules (i.e., declarative logical rules) that are compatible with MLN, and then execute these rules using spatial-aware statistical learning and inference algorithms. To show *Flash* effectiveness, we built three popular SPGMs: spatial Markov random fields (SMRF) [11], spatial hidden Markov models (SHMM) [12] and spatial Bayesian networks (SBN) [55]. Experimental evidence, based on three real spatial analysis applications, shows that *Flash* has at least two orders of magnitude speed up in learning the SPGM model parameters over state-of-the-art techniques.

Chapter 7

Conclusion

Spatial data has become ubiquitous everywhere, e.g., GPS data, satellite images, medical data, with increasingly sheer sizes. This raises the need for efficient spatial machine learning and analysis solutions to extract useful insights from such data. Meanwhile, Markov Logic Networks (MLN) have emerged as a powerful framework for building usable and scalable machine learning tools. Unfortunately, the MLN framework is ill-equipped for spatial applications because it ignores the distinguished spatial data characteristics. In this thesis, we showed how we adopt the MLN framework for big spatial data and applications. In particular, we focused on two orthogonal, but related, research directions.

The first research direction is to provide a native support for spatial data inside the MLN-based applications, and hence leads to a higher accuracy for such applications. As a case study, we introduced *Sya* (Chapter 3); the first spatial probabilistic knowledge base construction system. *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. It 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 knowledge base 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 of grounding and inference phases.

The second research direction is to exploit the MLN framework to scale up the performance of core spatial data analysis operations. In this direction, we proposed efficient MLN-based solutions for two important operations; *autologistic regression* (Chapter 4 and 5) and *spatial probabilistic graphical modeling* (Chapter 6), as follows:

- Autologistic regression is one of the most popular statistical tools to predict spatial phenomena in several applications including epidemic diseases detection, species occurrence prediction, earth observation and business management. In general, autologistic regression divides the space into a two-dimensional grid, where the prediction is performed at each cell in the grid. The prediction at any location is based on a set of predictors (i.e., features) at this location and predictions from neighboring locations. In this thesis, we address the problem of building efficient autologistic models with both binary and multinomial (i.e., categorical) prediction and predictor variables. Unfortunately, existing methods to build autologistic models are computationally expensive and do not scale up for large-scale grid data (e.g., fine-grained satellite images). Therefore, we introduced *TurboReg* and *RegRocket*, which are MLN-based systems to scale up the performance of binary and multinomial autologistic regression, respectively. These systems consider both the accuracy and efficiency aspects when learning the regression model parameters. They provide an equivalent representation of the prediction and predictor variables using MLN where the dependencies between these variables are transformed into first-order logic predicates. Then, they employ an efficient framework that learns the model parameters from the MLN representation in a distributed manner. Extensive experimental results based on two large real datasets show that *RegRocket* can build multinomial autologistic models, in minutes (i.e., almost three orders of magnitude faster than the best of existing techniques), for 1 million grid cells with 0.85 average F1-score.

- Spatial probabilistic graphical modeling is a crucial operation in many spatial analysis applications (e.g., meteorology, and environmental science). However, existing tools are neither generic nor scalable when dealing with big spatial data. Therefore, we introduced *Flash*; a framework for *generic* and *scalable* spatial data analysis. *Flash* exploits the expressiveness of MLN to represent the spatial probabilistic graphical models as a set of declarative logical rules. In addition, it provides spatial variations of the scalable

RDBMS-based learning and inference techniques of MLN to efficiently perform predictions. We have employed *Flash* to provide the MLN-based representation of three popular SPGM models including spatial Markov random fields (SMRF) [11], spatial hidden Markov models (SHMM) [12] and spatial Bayesian networks (SBN) [55]. Experimental evidence, based on three real spatial analysis applications, shows that *Flash* has at least two orders of magnitude speed up in learning the SPGM model parameters over state-of-the-art computational methods.

References

[1] Matthew Richardson and Pedro M. Domingos. Markov Logic Networks. *Machine Learning*, 62(1-2):107–136, 2006.

[2] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental Knowledge Base Construction Using DeepDive. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2015.

[3] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. HoloClean: Holistic Data Repairs with Probabilistic Inference. *VLDB Journal*, 10(11):1190–1201, August 2017.

[4] Shangpu Jiang, Daniel Lowd, and Dejing Dou. Learning to Refine an Automatically Extracted Knowledge Base Using Markov Logic. In *Proceedings of the IEEE International Conference on Data Mining, ICDM*, 2012.

[5] Making The Most Detailed Tweet Map Ever., 2014. `blog.mapbox.com/making-the-most-detailed-tweet-map-ever-b54da237c5ac`.

[6] Martin J. Wainwright and Michael I. Jordan. Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[7] Michael Wick, Andrew McCallum, and Gerome Miklau. Scalable Probabilistic Databases with Factor Graphs and MCMC. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2010.

[8] Ce Zhang and Christopher Ré. Towards High-throughput Gibbs Sampling at Scale: A Study Across Storage Managers. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2013.

[9] Christopher De Sa, Alex Ratner, Christopher Ré, Jaeho Shin, Feiran Wang, Sen Wu, and Ce Zhang. DeepDive: Declarative Knowledge Base Construction. *ACM SIGMOD Record*, 45(1):60–67, March 2016.

[10] John Hughes, Murali Haran, and Petruta C. Caragea. Autologistic Models for Binary Data on a Lattice. *Environmetrics*, 2011.

[11] Julian Besag. Spatial Interaction and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society*, 1974.

[12] Peter J. Green and Sylvia Richardson. Hidden Markov Models and Disease Mapping. pages 1055–1070, 2002.

[13] Marcio Pupin Mello, Bernardo Friedrich Theodor Rudorff, Marcos Adami, and Daniel Alves Aguiar. An R Implementation for Bayesian Networks Applied to Spatial Data. *Procedia Environmental Sciences*, 2011.

[14] Michael Genesereth and Nils Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1987.

[15] Telescope Hubbel site: Hubble Essentials: Quick Facts. `http://hubble.stsci.edu/the_telescope/hubble_essentials/`.

[16] European XFEL: The Data Challenge. `http://www.xfel.eu/news/2012/the_data_challenge/`.

[17] MODIS Land Products Quality Assurance Tutorial: Part1. `https://lpdaac.usgs.gov/sites/default/files/public/modis/docs/MODIS`.

[18] GnipBlog. Tag Archives: Geotagged Tweets. `https://blog.gnip.com/tag/geotagged-tweets/`.

[19] Twitter. The About webpage. `https://about.twitter.com/company`.

[20] Ibrahim Sabek and Mohamed F. Mokbel. Machine Learning Meets Big Spatial Data (Tutorial). In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, pages 1982–1985, 2019.

[21] Henry Makram. The Blue Brain Project. *Nature Reviews Neuroscience*, 7:153–160, 2006.

[22] Farhan Tauheed, Laurynas Biveinis, Thomas Heinis, Felix Schurmann, Henry Markram, and Anastasia Ailamaki. Accelerating Range Queries for Brain Simulations. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2012.

[23] L. Pickle, M. Szczur, D. Lewis, , and D. Stinchcomb. The Crossroads of GIS and Health Information: A Workshop on Developing a Research Agenda to Improve Cancer Control. *International Journal of Health Geographics*, 5(1):51, 2006.

[24] A. Auchincloss, S. Gebreab, C. Mair, and A. Diez Roux. A Review of Spatial Methods in Epidemiology: 2000-2010. *Annual Review of Public Health*, 33:107–22, April 2012.

[25] Y. Thomas, D. Richardson, and I. Cheung. Geography and Drug Addiction. *Springer Verlag*, 2009.

[26] Jamed Faghmous and Vipin Kumar. *Spatio-Temporal Data Mining for Climate Data: Advances, Challenges, and Opportunities*. Advances in Data Mining, Springer, 2013.

[27] James Faghmous, L. Styles, N. Gibson, and Vipin Kumar. Spatio-temporal Data Mining Methods for Ocean Eddy Monitoring. In *Proceeding of the Second International Workshop on Climate Informatics*, 2012.

[28] Zhe Jiang, Shashi Shekhar, Pradeep Mohan, James Knight, and J. Corcoran. Learning Spatial Decision Tree for Geographical Classification: A Summary of Results. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2012.

[29] K. Subbian and Arindam Banerjee. Climate Multi-Model Regression Using Spatial Smoothing. In *Proceedings of the SIAM International Conference on Data Mining, SDM*, 2013.

[30] Jagan Sankaranarayanan, Hanan Samet, Benjamin E. Teitler, Michael D. Lieberman, and Jon Sperling. TwitterStand: News in Tweets. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2009.

[31] Keras. `https://keras.io/`.

[32] Theano. `https://pypi.org/project/Theano/`.

[33] TensorFlow. `https://www.tensorflow.org/`.

[34] Caffe. `https://caffe.berkeleyvision.org/`.

[35] Pedro Domingos and Daniel Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan and Claypool Publishers, 2009.

[36] Yang Chen and Daisy Zhe Wang. Knowledge Expansion over Probabilistic Knowledge. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2014.

[37] Yang Chen, Xiaofeng Zhou, Kun Li, and Daisy Zhe Wang. Archimedes: Efficient Query Processing over Probabilistic Knowledge Bases. *ACM SIGMOD Record*, 46(2):30–35, September 2017.

[38] Parag Singla and Pedro Domingos. Entity Resolution with Markov Logic. In *Proceedings of the IEEE International Conference on Data Mining, ICDM*, 2006.

[39] Sebastian Riedel and Ivan Meza-Ruiz. Collective Semantic Role Labelling with Markov Logic. In *Proceedings of the Conference on Computational Natural Language Learning, CoNLL*, 2008.

[40] Céline Brouard, Christel Vrain, Julie Dubois, David Castel, Marie-Anne Debily, and Florence d'Alché Buc. Learning a Markov Logic Network for Supervised Gene Regulatory Network Inference. *BMC Bioinformatics*, 14(1):273, September 2013.

[41] Nikita A. Sakhanenko and David J. Galas. Markov Logic Networks in the Analysis of Genetic Data. *Journal of Computational Biology*, 17(11):1491–1508, November 2010.

[42] Ibrahim Sabek. Adopting Markov Logic Networks for Big Spatial Data and Applications. In *VLDB PhD Workshop*, 2019.

[43] Ce Zhang, Jaeho Shin, Christopher Ré, Michael Cafarella, and Feng Niu. Extracting Databases from Dark Data with DeepDive. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2016.

[44] Human Trafficking: Forbes. `www.forbes.com/sites/thomasbrewster/2015/04/17/darpa-nasa-and-partners-show-off-memex/`.

[45] Human Trafficking: Memex. `http://humantraffickingcenter.org/memex-helps-find-human-trafficking-cases-online/`.

[46] Ce Zhang, Vidhya Govindaraju, Jackson Borchardt, Tim Foltz, Christopher Ré, and Shanan Peters. GeoDeepDive: Statistical Inference Using Familiar Data-processing Languages. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2013.

[47] Shanan E. Peters, Ce Zhang, Miron Livny, and Christopher Ré. A Machine Reading System for Assembling Synthetic Paleontological Databases. *PLoS One*, 9(12), 2014.

[48] Julian Besag. Statistical Analysis of Non-Lattice Data. *Journal of the Royal Statistical Society*, 1975.

[49] Petruta C. Caragea and Mark S. Kaiser. Autologistic Models with Interpretable Parameters. *Journal of Agricultural, Biological, and Environmental Statistics*, 2009.

[50] Charles J. Geyer. On the Convergence of Monte Carlo Maximum Likelihood Calculations. *Journal of the Royal Statistical Society*, 1994.

[51] John Hughes. ngspatial: A Package for Fitting the Centered Autologistic and Sparse Spatial Generalized Linear Mixed Models for Areal Data. *The R Journal*, 2014.

[52] J. Moller, A. N. Pettitt, R. Reeves, and K. K. Berthelsen. An Efficient Markov Chain Monte Carlo Method for Distributions with Intractable Normalising Constants. *Biometrika*, 2006.

[53] Zilong Wang and Yanbing Zheng. Analysis of Binary Data via a Centered Spatial-temporal Autologistic Regression Model. *Journal of Environmental and Ecological Statistics*, 2013.

[54] Yanbing Zheng and Jun Zhu. Markov Chain Monte Carlo for a Spatial-Temporal Autologistic Regression Model. *Journal of Computational and Graphical Statistics*, 2008.

[55] bnspatial: Spatial Implementation of Bayesian Networks and Mapping. `https://ias.cs.tum.edu/dokuwiki/research/probcog`.

[56] Rafael Cano, Carmen Sordo, and José M. Gutiérrez. *Applications of Bayesian Networks in Meteorology*, pages 309–328. Springer, 2004.

[57] Stefano Balbi, Ferdinando Villa, Vahid Mojtahed, Karin Tessa Hegetschweiler, and Carlo Giupponi. A Spatial Bayesian Network Model to Assess the Benefits of Early Warning for Urban Flood Risk to People. *Natural Hazards and Earth System Sciences*, 2016.

[58] Julen Gonzalez-Redin, Sandra Luque, Laura Poggio, Ron Smith, and Alessandro Gimona. Spatial Bayesian Belief Networks as a Planning Decision Tool for Mapping Ecosystem Services Trade-offs on Forested Landscapes. *Environmental Research*, 2016.

[59] Ibrahim Sabek and Mohamed F. Mokbel. Sya: Enabling Spatial Awareness inside Probabilistic Knowledge Base Construction. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2020.

[60] Ibrahim Sabek et al. A Demonstration of Sya: A Spatial Probabilistic Knowledge Base Construction System. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2018.

[61] Ibrahim Sabek, Mashaal Musleh, and Mohamed F. Mokbel. TurboReg: A Framework for Scaling Up Spatial Logistic Regression Models. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2018.

[62] Ibrahim Sabek, Mashaal Musleh, and Mohamed F. Mokbel. RegRocket: Scalable Multinomial Autologistic Regression with Unordered Categorical Variables Using Markov

Logic Networks. *ACM Transactions on Spatial Algorithms and Systems*, 5(4):27:1–27:27, 2019.

[63] Ibrahim Sabek, Mashaal Musleh, and Mohamed F. Mokbel. Flash in Action: Scalable Spatial Data Analysis Using Markov Logic Networks. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2019.

[64] Ibrahim Sabek. Towards Scalable Spatial Probabilistic Graphical Modeling. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2019.

[65] Yang Chen and Daisy Zhe Wang. Web-Scale Knowledge Inference Using Markov Logic Networks. In *Proceedings of the ICML workshop on Structured Learning: Inferring Graphs from Structured and Unstructured Inputs*, 2013.

[66] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling Up Statistical Inference in Markov Logic Networks Using an RDBMS. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2011.

[67] Daisy Zhe Wang, Yang Chen, Sean Goldberg, Christan Grant, and Kun Li. Automatic Knowledge Base Construction Using Probabilistic Extraction, Deductive Reasoning, and Human Feedback. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, 2012.

[68] Christan Grant and Daisy Zhe Wang. A Challenge for Long-Term Knowledge Base Maintenance. *ACM Journal of Data and Information Quality, ACM JDIQ*, 6(2-3):7:1–7:3, June 2015.

[69] Xiaofeng Zhou, Yang Chen, and Daisy Zhe Wang. ArchimedesOne: Query Processing over Probabilistic Knowledge Bases. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2016.

[70] Robert Crane and Luke K. McDowell. Evaluating Markov Logic Networks for Collective Classification. In *Proceedings of the MLG Workshop at ACM SIGKDD*, 2011.

[71] Fei Wu and Daniel S. Weld. Automatically Refining the Wikipedia Infobox Ontology. In *Proceedings of the International Conference on World Wide Web, WWW*, 2008.

[72] Warren Shen, AnHai Doan, Jeffrey F. Naughton, and Raghu Ramakrishnan. Declarative Information Extraction Using Datalog with Embedded Extraction Predicates. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2007.

[73] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd edition, 2009.

[74] Tuyen N. Huynh and Raymond J. Mooney. Discriminative Structure and Parameter Learning for Markov Logic Networks. pages 416–423, 2008.

[75] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J. Smola. Parallelized Stochastic Gradient Descent. In *Proceedings of the Annual Conference on Neural Information Processing Systems, NIPS*, 2010.

[76] Soren Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the International Semantic Web Conference-Asian Semantic Web Conference*, 2007.

[77] Joanna Biega, Erdal Kuzey, and Fabian M. Suchanek. Inside YAGO2s: A Transparent Information Extraction Architecture. In *Proceedings of the International Conference on World Wide Web, WWW*, 2013.

[78] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2008.

[79] Omkar Deshpande, Digvijay S. Lamba, Michel Tourn, Sanjib Das, Sri Subramaniam, Anand Rajaraman, Venky Harinarayan, and AnHai Doan. Building, Maintaining, and Using Knowledge Bases: A Report from the Trenches. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2013.

[80] Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu. SystemT: A System for Declarative Information Extraction. *ACM SIGMOD Record*, 37(4):7–13, March 2009.

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

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

[83] Ranjit Bose. Knowledge Management-enabled Health Care Management Systems. *Expert Systems with Applications*, 24(1):59–71, 2003.

[84] Gjergji Kasneci, Maya Ramanath, Fabian Suchanek, and Gerhard Weikum. The YAGO-NAGA Approach to Knowledge Discovery. *ACM SIGMOD Record*, 37(4):41–47, March 2009.

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

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

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

[88] Mark S. Kaiser, Soumendra N. Lahiri, and Daniel J. Nordman. Goodness of Fit Tests for a Class of Markov Random Field Models. *The Annals of Statistics*, 2012.

[89] Texas Ground Water Database. http://www.twdb.texas.gov/groundwater/data/.

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

[91] Walid G. Aref and Hanan Samet. Efficient Processing of Window Queries in the Pyramid Data Structure. In *Proceedings of the ACM Symposium on Principles of Database Systems, PODS*, 1990.

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

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

[94] Luc Anselin, Ibnu Syabri, and Youngihn Kho. GeoDa: An Introduction to Spatial Data Analysis. *Geographical Analysis*, 38(1):5–22, 2006.

[95] W. R. Tobler. *Cellular Geography: Philosophy in Geography*. Springer, 1979.

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

[97] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized Search Trees for Database Systems. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 1995.

[98] Kun Li, Xiaofeng Zhou, Daisy Zhe Wang, Christan Grant, Alin Dobra, and Christopher Dudley. In-database Batch and Query-time Inference over Probabilistic Graphical Models using UDA-GIST. *VLDB Journal*, 26(2):177–201, April 2017.

[99] Ce Zhang and Christopher Ré. DimmWitted: A Study of Main-memory Statistical Analytics. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2014.

[100] Deepak Venugopal and Vibhav Gogate. On Lifting the Gibbs Sampling Algorithm. In *Proceedings of the Annual Conference on Neural Information Processing Systems, NIPS*, 2012.

[101] PostgreSQL. `https://www.postgresql.org/`.

[102] PostGIS. `http://postgis.net/`.

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

[104] Marti A. Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. In *Proceedings of the International Conference on Computational Linguistics, COLING*, 1992.

[105] Denny Vrandečić and Markus Krötzsch. Wikidata: A Free Collaborative Knowledge Base. *Communications of the ACM, CACM*, 57(10):78–85, 2014.

[106] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open Information Extraction from the Web. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, 2007.

[107] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an Architecture for Never-ending Language Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, 2010.

[108] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining, KDD*, 2014.

[109] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From Data Fusion to Knowledge Fusion. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2014.

[110] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale Information Extraction in KnowItAll: (Preliminary Results). In *Proceedings of the International Conference on World Wide Web, WWW*, 2004.

[111] Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, and Mausam Mausam. Open Information Extraction: The Second Generation. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, 2011.

[112] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A. Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. Building Watson: An Overview of the DeepQA Project. *AI MAGAZINE*, 31(3), 2010.

[113] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying Relations for Open Information Extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP*, 2011.

[114] Alexander Yates, Michael Cafarella, Michele Banko, Oren Etzioni, Matthew Broadhead, and Stephen Soderland. TextRunner: Open Information Extraction on the Web. In *Proceedings of the Annual Conference of the North American Association for Computational Linguistics, NAACL*, 2007.

[115] Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. Scalable Knowledge Harvesting with High Precision and High Recall. In *Proceedings of the ACM International Conference on Web Search and Data Mining, WSDM*, 2011.

[116] Fabian M. Suchanek, Mauro Sozio, and Gerhard Weikum. SOFIE: A Self-organizing Framework for Information Extraction. In *Proceedings of the International Conference on World Wide Web, WWW*, 2009.

[117] Hoifung Poon and Pedro Domingos. Joint Inference in Information Extraction. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, 2007.

[118] Dat Ba Nguyen, Abdalghani Abujabal, Nam Khanh Tran, Martin Theobald, and Gerhard Weikum. Query-driven On-the-fly Knowledge Base Construction. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2017.

[119] Jun Zhu, Zaiqing Nie, Xiaojiang Liu, Bo Zhang, and Ji-Rong Wen. StatSnowball: A Statistical Approach to Extracting Entity Relationships. In *Proceedings of the International Conference on World Wide Web, WWW*, 2009.

[120] Yang Chen, Daisy Zhe Wang, and Sean Goldberg. ScaLeKB: Scalable Learning and Inference over Large Knowledge Bases. *VLDB Journal*, 25(6):893–918, December 2016.

[121] Sen Wu, Luke Hsiao, Xiao Cheng, Braden Hancock, Theodoros Rekatsinas, Philip Levis, and Christopher Ré. Fonduer: Knowledge Base Construction from Richly Formatted Data. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2018.

[122] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. *Journal of Artificial Intelligence Research*, 194:28–61, January 2013.

[123] Bayu Distiawan Trisedya, Gerhard Weikum, Jianzhong Qi, and Ru Zhang. Neural Relation Extraction for Knowledge Base Enrichment. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, ACL*, 2019.

[124] Hao Xin, Rui Meng, and Lei Chen. Subjective Knowledge Base Construction Powered By Crowdsourcing and Knowledge Base. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2018.

[125] Gensheng Zhang, Damian Jimenez, and Chengkai Li. Maverick: Discovering Exceptional Facts from Knowledge Graphs. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2018.

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

[127] Max J. Egenhofer. Toward the Semantic Geospatial Web. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2002.

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

[129] Sören Auer, Jens Lehmann, and Sebastian Hellmann. LinkedGeoData: Adding a Spatial Dimension to the Web of Data. In *The Semantic Web*, 2009.

[130] Mordechai Haklay and Patrick Weber. OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive Computing*, 2008.

[131] Bayu Distiawan Trisedya, Jianzhong Qi, and Rui Zhang. Entity Alignment between Knowledge Graphs Using Attribute Embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, 2019.

[132] David Ferrucci and Adam Lally. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Journal of Natural Language Engineering*, 10(3-4):327–348, September 2004.

[133] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: An Architecture for Development of Robust HLT Applications. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, ACL*, 2002.

[134] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. A Brief Survey of Web Data Extraction Tools. *ACM SIGMOD Record*, 31(2):84–93, June 2002.

[135] Georg Gottlob, Christoph Koch, Robert Baumgartner, Marcus Herzog, and Sergio Flesca. The Lixto Data Extraction Project: Back and Forth Between Theory and Practice. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2004.

[136] Alchemy. `https://alchemy.cs.washington.edu/`.

[137] Parag Singla and Pedro Domingos. Lifted First-order Belief Propagation. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, 2008.

[138] Eric Gribkoff and Dan Suciu. SlimShot: In-database Probabilistic Inference for Knowledge Bases. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2016.

[139] Hoifung Poon and Pedro Domingos. Sound and Efficient Inference with Probabilistic and Deterministic Dependencies. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, 2006.

[140] Mark A. Wolters and C. B. Dean. Classification of Large-Scale Remote Sensing Images for Automatic Identification of Health Hazards: Smoke Detection Using an Autologistic Regression Classifier. *Statistics in Biosciences*, 2017.

[141] Nikos Koutsias. An Autologistic Regression Model for Increasing the Accuracy of Burned Surface Mapping using Landsat Thematic Mapper Data. *International Journal of Remote Sensing*, 2003.

[142] Daphne Lopez, M. Gunasekaran, and B. Senthil Murugan. Spatial Big Data Analytics of Influenza Epidemic in Vellore, India. In *Proceedings of the IEEE International Conference on Big Data, BigData*, 2014.

[143] Michael Sherman, Tatiyana V. Apanasovich, and Raymond J. Carroll. On Estimation in Binary Autologistic Spatial Models. *Journal of Statistical Computation and Simulation*, 2006.

[144] Juan Ferrandiz, Antonio Lopez, Agustin Llopis, Maria Morales, and Maria Luisa Tejerizo. Spatial Interaction between Neighbouring Counties: Cancer Mortality Data in Valencia (Spain). *Biometrics*, 1995.

[145] R. Sanderson, M. D. Eyre, S. P. Rushton, and Kaj Sand-Jensen. Distribution of Selected Macroinvertebrates in a Mosaic of Temporary and Permanent Freshwater Ponds as Explained by Autologistic Models. *Ecography*, 2005.

[146] Nathalie Augustin, Moira A. Mugglestone, and Stephen T. Buckland. An Autologistic Model for the Spatial Distribution of Wildlife. *Journal of Applied Ecology*, 1996.

[147] Fangliang He, Julie Zhou, and Hongtu Zhu. Autologistic Regression Model for the Distribution of Vegetation. *Journal of Agricultural, Biological, and Environmental Statistics*, 2003.

[148] Marcia L. Gumpertz, Jonathan M. Graham, and Jean B. Ristaino. Autologistic Model of Spatial Pattern of Phytophthora Epidemic in Bell Pepper: Effects of Soil Variables on Disease Presence. *Journal of Agricultural, Biological, and Environmental Statistics*, 1997.

[149] Sangkil Moon and Gary J. Russell. Predicting Product Purchase from Inferred Customer Similarity: An Autologistic Model Approach. *Management Science*, 2008.

[150] Brian L. Sullivan, Christopher L. Wood, Marshall J. Iliff, Rick E. Bonney, Daniel Fink, and Steve Kelling. eBird: A Citizen-based Bird Observation Network in the Biological Sciences. *Biological Conservation*, 2009.

[151] David R. Cox. The Regression Analysis of Binary Sequences (with discussion). *Journal of the Royal Statistical Society*, 1958.

[152] Colin M. Beale, Jack J. Lennon, Jon M. Yearsley, Mark J. Brewer, and David A. Elston. Regression Analysis of Spatial Data. *Ecology Letters*, 2010.

[153] NASA EarthData. https://earthdata.nasa.gov/earth-observation-data.

[154] Catherine Linard and Andrew J. Tatem. Large-scale Spatial Population Databases in Infectious Disease Research. *International Journal of Health Geographics*, 2012.

[155] J. Michael Scott, Patricia J. Heglund, and Michael L. Morrison et al. Predicting Species Occurrences: Issues of Accuracy and Scale. *Journal of Mammalogy*, 2002.

[156] Tjelmeland Hakon and Besag Julian. Markov Random Fields with Higher-order Interactions. *Scandinavian Journal of Statistics*, 1998.

[157] R.A. Finkel and J.L. Bentley. Quad Trees a Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 1974.

[158] J. Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems, TODS*, 1984.

[159] OpenDataMinneapolis. http://opendata.minneapolismn.gov/.

[160] MinnesotaCompass. http://www.mncompass.org/.

[161] Jun Zhu, Hsin-Cheng Huang, and Jungpin Wu. Modeling Spatial-temporal Binary Data using Markov Random Fields. *Journal of Agricultural, Biological, and Environmental Statistics*, 2005.

[162] Anne Gégout-Petit and Shuxian Li. Two-step Centered Spatio-temporal Auto-logistic Regression Model. In *Applied Statistics for Development in Africa, SADA*, 2016.

[163] Mark A. Wolters. Better Autologistic Regression. *Frontiers in Applied Mathematics and Statistics*, 3:24, 2017.

[164] Brenda Betancourt, Abel Rodríguez, and Naomi Boyd. Investigating Competition in Financial Markets: A Sparse Autologistic Model for Dynamic Network Data. *Journal of Applied Statistics*, 45(7):1157–1172, 2018.

[165] Murali Haran. *Gaussian Random Field Models for Spatial Data*, pages 449–478. Chapman and Hall/CRC, 2011.

[166] C. A. Gotway and W. W. Stroup. A Generalized Linear Model Approach to Spatial Data Analysis and Prediction. *Journal of Agricultural, Biological, and Environmental Statistics*, 1997.

[167] Havard Rue and Leonhard Held. *Gaussian Markov Random Fields: Theory And Applications (Monographs on Statistics and Applied Probability)*. Chapman & Hall/CRC, 2005.

[168] Jennifer A. Hoeting, Molly Leecaster, and David Bowden. An Improved Model for Spatially Correlated Binary Responses. *Journal of Agricultural, Biological, and Environmental Statistics*, 1999.

[169] T. Nguyen, I. Hettiarachchi, A. Khatami, L. Gordon-Brown, C. P. Lim, and S. Nahavandi. Classification of Multi-Class BCI Data by Common Spatial Pattern and Fuzzy System. *IEEE Access*, 6:27873–27884, 2018.

[170] K. Tangthaikwan, N. Keeratipranon, and A. Agsornintara. Multiclass Support Vector Machine for Classification Spatial Data from Satellite Image. In *Proceedings of the International Conference on Knowledge and Smart Technology, KST*, pages 111–115, 2017.

[171] USGS National Land Cover Dataset (NLCD). `catalog.data.gov/dataset/usgs-national-land-cover-dataset-nlcd-downloadable-data-collection`.

[172] Multi-Resolution Land Cover Characteristics (MRLC) Consortium. `https://www.mrlc.gov/data`.

[173] J. Engel. Polytomous Logistic Regression. *Statistica Neerlandica*, 42(4):233–252, 1988.

[174] Stan Z. Li. *Markov Random Field Modeling in Image Analysis*. Springer, 3rd edition, 2009.

[175] J. Li, J. M. Bioucas-Dias, and A. Plaza. Spectral-Spatial Hyperspectral Image Segmentation Using Subspace Multinomial Logistic Regression and Markov Random Fields. *IEEE Transactions on Geoscience and Remote Sensing*, 50(3):809–823, 2012.

[176] Parag Singla and Pedro Domingos. Discriminative Training of Markov Logic Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, pages 868–873, 2005.

[177] Amin Tayyebi, Mahmoud Reza Delavar, Mohammad Javad Yazdanpanah, Bryan Christopher Pijanowski, Sara Saeedi, and Amir Hossein Tayyebi. A Spatial Logistic Regression Model for Simulating Land Use Patterns: A Case Study of the Shiraz Metropolitan Area of Iran. In *Advances in Earth Observation of Global Change*, 2010.

[178] Digital Elevation Model (DEM) of Minnesota. `http://www.mngeo.state.mn.us/chouse/metadata/dem24ras.html`.

[179] USGS National Transportation Dataset (NTD). `catalog.data.gov/dataset/usgs-national-transportation-dataset-ntd-downloadable-data-collectionde`

[180] Shashi Shekhar, Michael R. Evans, James M. Kang, and Pradeep Mohan. Identifying Patterns in Spatial Information: A Survey of Methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2011.

[181] Mao Xi, Li Qi, Zhang Zimin, and Zhu Qiang. Application of Spatial Information Search Engine Based on Ontology in Public Health Emergence. In *Proceedings of the International Conference on Bioinformatics and Biomedical Engineering, ICBBE*, 2009.

[182] Michael D. M. Bader. GIS and Public Health. *Spatial Demography*, 2013.

[183] Shashi Shekhar, Toneluh A. Yang, and Peter A. Hancock. An Intelligent Vehicle Highway Information Management System. *Computer-Aided Civil and Infrastructure Engineering*, 8(3):175–198, May 1993.

[184] Jean-Claude Thill. Geographic Information Systems For Transportation in Perspective. *Journal of Transportation Research Part C: Emerging Technologies*, 2000.

[185] M. Kanevski, R. Parkin, A. Pozdnukhov, V. Timonin, M. Maignan, V. Demyanov, and S. Canu. Environmental Data Mining and Modeling Based on Machine Learning Algorithms and Geostatistics. *Journal of Environmental Modelling and Software*, 19(9):845–855, 2004.

[186] James Faghmous, Matthew Le, Muhammed Uluyol, Vipin Kumar, and Snigdhansu Chatterjee. A Parameter-Free Spatio-Temporal Pattern Mining Model to Catalog Global Ocean Dynamics. In *Proceedings of the IEEE International Conference on Data Mining, ICDM*, 2013.

[187] Trang VoPham, Jaime E. Hart, Francine Laden, and Yao-Yi Chiang. Emerging Trends in Geospatial Artificial Intelligence (geoAI): Potential Applications for Environmental Epidemiology. *Environmental Health*, 2018.

[188] P. Stolorz, H. Nakamura, E. Mesrobian, R. R. Muntz, E. C. Shek, J. R. Santos, J. Yi, K. Ng, S.-Y. Chien, H. Nakamura, C. R. Mechoso, and J. D. Farrara. Fast Spatio-temporal Data Mining of Large Geophysical Datasets. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining, KDD*, 1995.

[189] Auroop R. Ganguly and Karsten Steinhaeuser. Data Mining for Climate Change and Impacts. In *Proceedings of the IEEE International Conference on Data Mining Workshops*, 2008.

[190] James H. Faghmous and Vipin Kumar. *Spatio-temporal Data Mining for Climate Data: Advances, Challenges, and Opportunities*, pages 83–116. Springer, 2014.

[191] Brian E. Mennecke. Understanding the Role of Geographic Information Technologies in Business: Applications and Research Directions. *Journal of Geographic Information and Decision Analysis*, 1997.

[192] Robert Tibshirani and Pei Wang. Spatial Smoothing and Hot Spot Detection for CGH Data Using the Fused Lasso. *Biostatistics*, 9(1):18–29, January 2008.

[193] Jyoti Singh et al. Event Classification and Location Prediction from Tweets During Disasters. *Annals of Operations Research*, 2017.

[194] Andrew O. Finley, Sudipto Banerjee, and Bradley P. Carlin. spBayes: An R Package for Univariate and Multivariate Hierarchical Point-Referenced Spatial Models. *Journal of Statistical Software*, 19(4):1–24, April 2007.

[195] Frank Warmerdam. *The Geospatial Data Abstraction Library*, pages 87–104. Springer Berlin Heidelberg, 2008.

[196] Sergio J. Rey and Luc Anselin. *PySAL: A Python Library of Spatial Analytical Methods*, pages 175–193. Springer Berlin Heidelberg, 2010.

[197] Ned Levine. *CrimeStat: A Spatial Statistical Program for the Analysis of Crime Incidents*, pages 381–388. Springer International Publishing, 2017.

[198] ESRI ArcGIS. `https://www.esri.com/en-us/arcgis/about-arcgis/overview`.

[199] Michael J. de Smith, Michael F. Goodchild, and Paul A. Longley. *Geospatial Analysis: A Comprehensive Guide to Principles, Techniques and Software Tools*. Troubador Publishing, 2018.

[200] shmm: An R Implementation of Spatial Hidden Markov Models. `github.com/mawp/shmm`, 2019.

[201] Sanjay Chawla, Shashi Shekhar, Weili Wu, and Uygar Ozesmi. *Modeling Spatial Dependencies for Mining Geospatial Data*. 2001.

[202] Chicago Crime Data, 2019. `data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2/`.

[203] Havard Rue and Hakon Tjelmeland. Fitting Gaussian Markov Random Fields to Gaussian Fields. *Scandinavian Journal of Statistics*, 29(1):31–49, 2002.

[204] Dan Cornford, Lehel Csató, and Manfred Opper. Sequential, Bayesian Geostatistics: A Principled Method for Large Data Sets. *Geographical Analysis*, 37(2):183–199, 2005.

[205] Michael Stein, Zhiyi Chi, and Leah Welty. Approximating Likelihoods for Large Spatial Data Sets. *Journal of the Royal Statistical Society*, 66(2):275–296, 2004.

[206] Sudipto Banerjee, Alan Gelfand, Andrew Finley, and Huiyan Sang. Gaussian Predictive Process Models for Large Spatial Data Sets. *Journal of the Royal Statistical Society*, 70(4):825–848, 2008.

[207] Luc Anselin and Sergio J. Rey. Spatial Econometrics in an Age of CyberGIScience. *International Journal of Geographical Information Science*, 2012.

[208] Golam Kabir, Solomon Tesfamariam, Alex Francisque, and Rehan Sadiq. Evaluating Risk of Water Mains Failure using a Bayesian Belief Network Model. *European Journal of Operational Research*, 2015.

[209] David N. Barton, Tor Haakon Bakken, and Anders L. Madsen. Using a Bayesian Belief Network to Diagnose Significant Adverse Effect of the EU Water Framework Directive on Hydropower Production in Norway. *Journal of Applied Water Engineering and Research*, 2016.

[210] Dries Landuyt, Steven Broekx, Rob D'hondt, Guy Engelen, Joris Aertsens, and Peter L. M. Goethals. A Review of Bayesian Belief Networks in Ecosystem Service Modelling. *Environmental Modelling and Software*, 2013.

[211] Carl S. Smith, Alison L. Howes, Bronwyn Price, and Clive A. McAlpine. Using a Bayesian Belief Network to Predict Suitable Habitat of an Endangered Mammal – The Julia Creek Dunnart (Sminthopsis Douglasi). *Biological Conservation*, 2007.

[212] Finn V. Jensen. *Introduction to Bayesian Networks*. Springer-Verlag, 1996.

[213] Dominik Jain, Klaus von Gleissenthall, and Michael Beetz. Bayesian Logic Networks and the Search for Samples with Backward Simulation and Abstract Constraint Learning. In *KI 2011: Advances in Artificial Intelligence*, 2011.

[214] ProbCog: A Toolbox for Statistical Relational Learning and Reasoning. `https://ias.cs.tum.edu/dokuwiki/research/probcog`.

[215] Roberto Ambrosini et al. Modelling the Progression of Bird Migration with Conditional Autoregressive Models Applied to Ringing Data. *PLOS ONE*, 9(7):1–10, 2014.

[216] Francesco Bartolucci et al. A Latent Markov Model for Detecting Patterns of Criminal Activity. *Journal of the Royal Statistical Society*, 170(1):115–132, 2007.