TECHNICAL UNIVERSITY OF CRETE

MASTER THESIS

Online Structure Learning for Markov Logic Networks using Background Knowledge Axiomatization

Author:

Evangelos MICHELIOUDAKIS

Supervisor:

Assoc. Prof. Michail G. LAGOUDAKIS

THESIS COMMITTEE

Assoc. Prof. Michail G. LAGOUDAKIS (ECE)

Professor Minos GAROFALAKIS (ECE)

Dr. Alexander ARTIKIS (University of Piraeus, NCSR Demokritos)

Technical University of Crete

School of Electronic and Computer Engineering

in collaboration with the

National Centre for Scientific Research "Demokritos"

Institute of Informatics and Telecommunications

April 2016

“The question of whether computers can think
is like the question of whether submarines can swim.”
— Edsger W. Dijkstra

Abstract

Many domains of interest today are characterized by both uncertainty and complex relational structure. Therefore, probabilistic structure learning is a popular research topic in artificial intelligence and machine learning. The research area of Statistical Relational Learning (SRL) specifically attempts to effectively represent, reason, and learn in domains that are governed by these characteristics. This thesis studies the problem of probabilistic structure learning under the Markov Logic Networks (MLN) framework. In particular, it addresses the issue of exploiting background knowledge axiomatization to effectively constrain the space of possible structures by learning clauses subject to specific characteristics defined by these axioms. We focus on the domain of symbolic event recognition under uncertainty by using the axiomatization of a probabilistic variant of the Event Calculus (`MLN-EC`) as background knowledge. We employ an online strategy in order to effectively handle large training sets and incrementally refine the previously learned structure. We demonstrate the effectiveness of our method through experiments in the domain of activity recognition, using a publicly available benchmark dataset for video surveillance.

Περίληψη

Πολλά ενδιαφέροντα προβλήματα σήμερα χαρακτηρίζονται τόσο από αβεβαιότητα όσο και από περίπλοκη σχεσιακή δομή. Ως εκ τούτου, η πιθανοτική μάθηση σχεσιακής δομής είναι ένα δημοφιλές θέμα έρευνας στον τομέα της τεχνητής νοημοσύνης και της μηχανικής μάθησης. Η περιοχή έρευνας της Στατιστικής Σχεσιακής Μάθησης (Statistical Relational Learning) επιχειρεί να ανακαλύψει τρόπους για αποτελεσματική αναπαράσταση, πιθανοτικό συμπερασμό, και μηχανική μάθηση σε προβλήματα που διέπονται από αυτά τα χαρακτηριστικά. Αυτή η διατριβή μελετά το πρόβλημα της πιθανοτικής μάθησης σχεσιακής δομής υπό την σκοπιά των Μαρκωβιανών Λογικών Δικτύων (Markov Logic Networks). Ειδικότερα, εξετάζει το ζήτημα της αξιοποίησης αξιωμάτων που προϋπάρχουν ως γνωστικό υπόβαθρο ώστε να περιορίσει αποτελεσματικά το χώρο των πιθανών δομών μαθαίνοντας κανόνες που υπόκεινται σε ειδικά χαρακτηριστικά που ορίζουν αυτά τα αξιώματα. Επικεντρωνόμαστε στην περιοχή της συμβολικής αναγνώρισης γεγονότων υπό συνθήκες αβεβαιότητας, χρησιμοποιώντας τα αξιώματα που ορίζονται από μια πιθανοτική παραλλαγή του Λογισμού Συμβάντων (MLN–EC) ως γνωστικό υπόβαθρο. Χρησιμοποιούμε μια σταδιακή στρατηγική, προκειμένου να χειριστούμε αποτελεσματικά τα μεγάλα σύνολα εκπαίδευσης και να βελτιώσουμε σταδιακά την δομή σε κάθε βήμα της διαδικασίας. Αποδείκνυουμε την αποτελεσματικότητα της μεθόδου μας μέσα από πειράματα στον τομέα της αναγνώρισης ανθρώπινων δραστηριοτήτων, χρησιμοποιώντας ως μέτρο σύγκρισης ένα διαθέσιμο στο κοινό σύνολο δεδομένων από βιντεοεπιτήρηση.

Acknowledgements

First and foremost, I would like to thank my advisors at NCSR Demokritos Dr. Georgios Paliouras and Dr. Alexander Artikis for their guidance and limitless encouragement during the course of my work. I really appreciate all the valuable discussions, the ideas, the willingness to share their extensive knowledge, and the patience they showed in my mistakes. Their help has been proved indispensable. I am also deeply thankful to my supervisor at TUC Michail G. Lagoudakis for the trust that he placed on me and made this work possible by giving me the opportunity to enstablish a collaboration with NCSR Demokritos and obtain such a great experience.

I am also very grateful to my colleague Dr. Anastasios Skarlatidis for his continuous support and invaluable assistance. I wish to thank him for his inspiration and the motivating discussions we held throughout our collaboration. He taught me in depth many of the concepts governing the field of Statistical Relational Learning, as well as how to become a better software engineer. It was a privilege to work with him and to benefit from his knowledge and friendship. I surely own the quality of this work to him.

Special thanks to Konstantinos Pechlivanis, Alexandros Mavrommatis, Stella Maropaki, Sotiris Sourlantzis, Giannis Liverios, Alexandros Armaos, Antonis Kechribaris, Stelios Christinakis, Panagiotis Kazantzas, Vasilis Thanopoulos, and Eleni Xynogala for their valuable friendship, for the beautiful moments we spent together and for always being there for me.

Last, but by no means least, I would like to thank my parents for everything they have done over these years. I would like to dedicate this work to them. They were always supportive and encouraging either to my successes or failures. Thank you.

Evangelos Michelioudakis
Chania, April 2016

To my parents.

Contents

List of Figures

List of Tables

1 Introduction

"The only source of knowledge is experience."
— Albert Einstein

Many real-world application domains are characterized by both uncertainty and complex relational structure. For instance, in social network analysis the individuals are related to one another via friendship or collaborations; in computational biology, one is usually interested in relations or interactions between chemical substances; in natural language processing tasks, it its often necessary to reason about the relationships of documents or tokens within a sentence; in activity recognition relations are defined over entities of persons or objects. Furthermore, applications such the ones mentioned above almost always contain noisy data due their real-world nature. Regularities in these domains are very hard to identify manually, and thus automatically learning them from data is desirable. Therefore, methods for unifying the strengths of logic and probabilistic modeling have become an important aspect of recent research in the area of machine learning. In particular, the fields of Probabilistic Inductive Logic Programming (PILP) and Statistical Relational Learning (SRL) concern the induction of probabilistic knowledge by combining the power of logic and probability and adopting principles of Inductive Logic Programming (ILP) and statistical learning. Furthermore, the collection and processing of data from various uncorrelated sources, which is common nowadays, provides a significant resource for knowledge discovery. In particular, it facilitates the automated discovery of multi-relational dependencies over these data useful for improving business process and providing services of higher quality.

The data provided for such a SRL system in many occasions represent temporal activities, that can be represented by *events*. Therefore, the multi-relational dependencies discovered by the machine learning procedure can be further used for the accurate detection of significant events by employing Complex Event Processing (CEP) techniques. CEP techniques recognize *composite events* (CEs) of interest, based on input streams of *simple derived events* (SDEs). CEs are defined as relational structures over other sub-events, either CEs or SDEs. Such CE definitions take the form of rules, usually expressed in a

formal language, that capture the knowledge of domain experts. Due to the dynamic nature of the real-world applications, the CE definitions may need to be refined over time or the current knowledge base may need to be enhanced with new definitions. Manual creation of event definitions is a tedious and cumbersome process and thus machine learning techniques to automatically derive the definitions are essential.

1.1 Motivation

One of the logic-based representations that handles uncertainty, proposed in the area of SRL, is Markov Logic which combines finite first-order logic and probabilistic graphical models. Structure learning approaches that focus on this formalism have been successfully applied to a variety of applications where uncertainty holds. However, most of these methods are batch algorithms that cannot handle large training sets. Moreover, these batch learning methods are data-driven and must repeatedly perform inference over the entire training set in each learning iteration, which becomes computationally expensive, and renders them unusable to real-world applications.

Huynh and Mooney [2011b] proposed an online strategy based on Markov Logic for updating the structure of the model in order to effectively handle large training datasets. Based on the model's incorrect predictions the proposed approach searches for relational dependencies that help enhancing these predictions. Nevertheless, it does not exploit background knowledge during the search procedure and may spend a lot of time exploring structures that are very common and therefore largely useless for the purposes of learning, yielding models that are not adequate generalizations.

Therefore, the approach proposed in Huynh and Mooney [2011b] is not suitable in a number of real-world event recognition tasks, such as activity recognition, because it cannot effectively handle a search space over a very large domain of constants (e.g. time), leading to an explosion of relational dependencies. There are also other large domain settings that cannot be handled effectively by existing methods, such as social network analysis and many tasks in natural language processing.

The motivation behind this thesis is the potential benefit of exploiting background knowledge in order to effectively constrain the search space of possible structures during learning. The space can be constrained subject to specific characteristics introduced by the rules governing a specific task that are usually stated as axioms. In our case the rules are the domain-independent axioms of the Event Calculus which is used for Complex Event Recognition.

1.2 Thesis Contribution

We focus on learning the structure of a logic-based model in situations where various forms of uncertainty hold, in order to be able to apply efficient and accurate event recognition. Moreover, we handle large training sets during the learning procedure by employing an online strategy. We propose a probabilistic structure learning method called OSLα that exploits background knowledge in order to efficiently search the space of possible structures, yielding an accurate model. To demonstrate the benefits of the proposed approach, the method is evaluated on human activity recognition.

OSLα extends the approach of [Huynh and Mooney, 2011b] by using the axiomatization of a probabilistic variant of the Event Calculus called MLN–EC, in order to define templates over the space of possible structures leading into an efficient template-guided search procedure. In principle, any axiomatization having similar properties can be used to constraint the search space of a similar problem. The proposed method can handle a simple form of first-order logic functions having finite domains and learn rules using these functions.

1.3 Thesis Outline

Chapter 2 provides the required background for the Event Calculus, Statistical Relational Learning, Markov Logic Networks, MLN–EC and the open-source software LoMRF. Furthermore, it also presents and discusses related work on structure learning. Chapter 3 describes the proposed method (OSLα) for Online Structure Learning by exploiting the background knowledge axiomatization. In Chapter 4, we present the experimental evaluation results of OSLα using a publicly available benchmark dataset for human activity recognition. Finally, in Chapter 5, we propose directions for future work regarding open issues and conclude.

2 Background

"The beginning is the most important part of the work."
— Plato

We focus on learning the structure of a logic-based model in situations where various forms of uncertainty hold, in order to be able to apply efficient and accurate event recognition. Moreover, we focus on employing an online learning strategy in order be able to exploit large datasets during the structure learning procedure. This chapter provides the required background for our work. The proposed method of this work is based both on the `MLN-EC` method, a variant of the Event Calculus formalism which handles uncertainty by employing a Statistical Relational Learning framework, as well as on a previously developed online structure learning algorithm [Huynh and Mooney, 2011b]. In Section 2.1 we briefly present the Event Calculus formalism. Then in Section 2.2, we present a state-of-the-art Statistical Relational Learning framework; that is Markov Logic Networks along with the `MLN-EC` method, as well as inference, weight learning methods, and LoMRF; an open-source implementation of Markov Logic Networks that we contributed our work. Finally, in Section 2.3, we present related work in the domain of probabilistic structure learning for Markov Logic Networks.

2.1 The Event Calculus

The Event Calculus, originally introduced by Kowalski and Sergot [1986], is a first-order predicate calculus for reasoning about events and their effects. A number of different variations of the original formalism have been proposed using either logic programming or classical logic — see Shanahan [1999], Miller and Shanahan [2002] and Mueller [2008] for surveys. Most of these dialects share the same ontology and core domain-independent axioms. The ontology consists of *time-points*, *events* and *fluents*. The underlying time model is often linear and may represent time-points as real or integer numbers. A *fluent* is a property whose value may change over time. When an *event* occurs it may change the value of a fluent. The core domain-independent axioms define

whether a fluent holds or not at a specific time-point. In addition, they also incorporate the common sense *law of inertia*, according to which fluents persist over time, unless they are affected by an event occurrence.

We consider finite domains of time-points, events and fluents, that are represented by the sets $\mathcal{T}$, $\mathcal{E}$ and $\mathcal{F}$, respectively. Moreover, individual entities appearing in a particular event recognition task, e.g., persons, objects, activities etc., are represented by constants in the finite set $\mathcal{O}$. Among the many variants of the original Event Calculus formalism, we chose a probabilistic version of the Event Calculus (MLN–EC) proposed by Skarlatidis et al. [2015] that is expressed in first-order logic and modeled using Markov Logic Networks. MLN–EC is based on the Discrete Event Calculus and constitutes a probabilistic variant of the formalism, which is desirable for the purpose of our work.

The Discrete Event Calculus[1] (DEC) is an alternative formulation of Shanahan's Full Event Calculus [Shanahan, 1999], which has been proved to be logically equivalent to the latter one when the domain of time-points is limited to integers [Mueller, 2008] — i.e., $\mathcal{T} \in \mathbb{Z}$. The original DEC is composed of twelve domain-independent axioms. On the contrary, MLN–EC is focused only on a subset of the domain-independent axioms that determine the influence of the events to fluents as well as the inertia of fluents. More details are given in Section 2.2.2.

2.2 Statistical Relational Learning

The Event Calculus can compactly represent complex event relations. A knowledge base of Event Calculus axioms and CE definitions is defined by a set of first-order logic formulas. Each formula is composed of predicates that associate variables or constants, representing SDEs, CEs, time-points, etc. One of the strong motivations for using such a relational representation is its ability to directly express dependencies between related instances — e.g., events. While this type of representation is highly expressive, it cannot handle uncertainty. Each formula imposes a (hard) constraint over the set of possible worlds, that is, Herbrand interpretations. Therefore, a missed or an erroneous SDE detection can have a significant effect on the event recognition results. For example, an initiation of a specific CE may be based on an erroneously detected SDE, thus causing the recognition of a CE to be certain.

Statistical machine learning systems, e.g., methods that are based on probabilistic graphical models [Lafferty et al., 2001; Murphy, 2002; Rabiner and Juang, 1986], adopt a probabilistic approach to handle uncertainty. Such probabilistic models have been successfully applied in many real-word applications that involve various forms of uncertainty, such as speech recognition, natural language processing, activity recognition, etc. By

[1] An open-source implementation of the Discrete Event Calculus is available at http://decreasoner.sourceforge.net

employing statistical learning techniques, the parameters of such models can be estimated automatically from training data. Compared to logic-based methods, probabilistic methods are less flexible for modeling several entities and relations among them. Thus, by relying on propositional representations, it is difficult to represent complex relational data or incorporate prior domain knowledge (e.g., knowledge from experts or common sense knowledge). Consequently, these methods are usually specifically extended for modeling particular applications.

Statistical Relational Learning (SRL) aims to develop methods that can effectively represent, reason and learn in domains with uncertainty and complex relational structure (e.g., relations among instances of SDEs and CEs). As shown in Figure 2.1, SRL combines a logic-based representation with probabilistic modeling and machine learning. In the domain of event recognition, the logic-based representation allows one to naturally define the relations between events and incorporate existing domain knowledge. This particularly expressive representation is combined with probabilistic modeling, in order to naturally handle uncertainty. Using machine learning techniques, the model can be automatically estimated or refined according to the given set of example data.

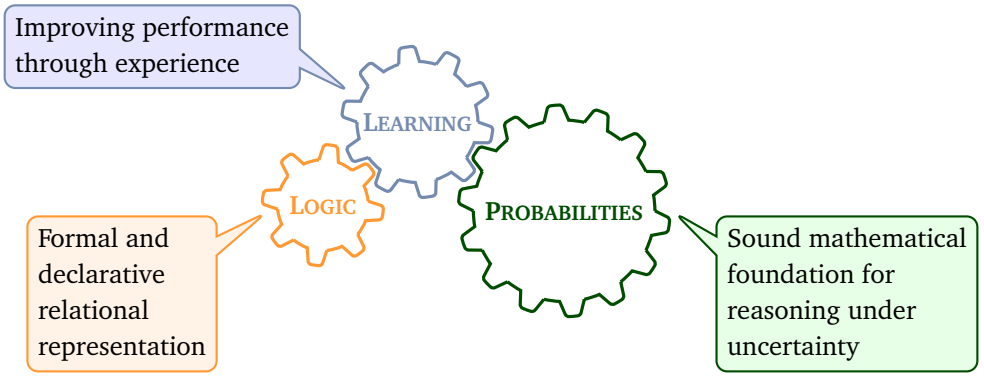

FIGURE 2.1: Statistical Relational Learning combines logic-based representation with probabilistic modeling and machine learning (after [Skarlatidis, 2014]).

2.2.1 Markov Logic Networks

Markov Logic Networks (MLNs) [Domingos and Lowd, 2009] is a state-of-the-art SRL method that provides a framework which combines first-order logic representation with Markov Network modeling. The basic idea is to soften the constraints that are imposed by the formulas of a knowledge base and be able to perform probabilistic inference. Each formula F_i is represented in first-order logic and is accompanied by a weight value $w_i \in \mathbb{R}$ that reflects its confidence. The higher the value of weight w_i, the stronger the constraint represented by formula F_i. Therefore, unlike classical logic, all worlds in MLNs are possible and they are quantified by a certain probability. The probability of a world increases as the number of formulas it violates decreases, other things being equal.

Consequently, a knowledge base in MLNs may contain both hard and soft-constrained formulas. Hard-constrained formulas are always accompanied by an infinite weight value indicating that the captured knowledge is certain. Therefore, an acceptable world (i.e., a possible Herbrand interpretation) must at least satisfy the hard constraints. On the contrary, soft constraints capture imperfect knowledge, allowing for the existence of worlds in which the knowledge is violated.

Formally, a knowledge base L of weighted formulas, together with a finite domain of constants $\mathcal{C}$, define a ground Markov network $M_{L,\mathcal{C}}$. All formulas are converted into *clausal normal form* (CNF) and each one of them is grounded according to the domain of its distinct first-order logic variables, e.g., a variable over the domain of time-points $\mathcal{T}$. The nodes in the resulting $M_{L,\mathcal{C}}$ are Boolean random variables, also called ground atoms, each one corresponding to a possible grounding of a predicate that appears in L. The predicates of ground clauses form cliques in $M_{L,\mathcal{C}}$ and each clique is accompanied by the weight w_i of the corresponding formula F_i in L. Moreover, each clique defines a Boolean *feature*, taking the value 1 when the ground clause is true and 0 otherwise. $M_{L,\mathcal{C}}$ defines a probability distribution over possible worlds and is represented as a log-linear model.

The aim of event recognition is to recognize CEs of interest given a stream of SDEs. Consequently, the focus is on discriminative MLNs [Singla and Domingos, 2005], that are related to Conditional Random Fields [Lafferty et al., 2001; Sutton and McCallum, 2007]. Boolean random variables in $M_{L,\mathcal{C}}$ are partitioned into a set of evidence random variables X, and a set of query random variables Y. The former correspond to the input SDEs (e.g. `HappensAt` predicates) while the latter correspond to the CEs of interest (e.g. `HoldsAt` predicates). The joint probability distribution of a possible assignment of $Y{=}\mathbf{y}$, conditioned over a given assignment of $X{=}\mathbf{x}$, is defined as follows:

$$P(Y{=}\mathbf{y} \mid X{=}\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{i=1}^{|F_c|} w_i n_i(\mathbf{x}, \mathbf{y})\right) \tag{2.1}$$

Vectors $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$ represent a possible assignment to evidence X and query/hidden variables Y, respectively. $\mathcal{X}$ and $\mathcal{Y}$ are the sets of all possible assignments that the evidence X and query/hidden variables Y can take. F_c is the set of clauses produced by the knowledge base L and the constants in $\mathcal{C}$. The scalar value w_i is the weight of the i-th clause and feature $n_i(\mathbf{x}, \mathbf{y})$ is the number of satisfied groundings of the i-th clause in $\mathbf{x}$ and $\mathbf{y}$. $Z(\mathbf{x})$ is called partition function and normalizes the probability over all possible assignments $\mathbf{y}' \in \mathcal{Y}$ of query/hidden variables given the assignment $\mathbf{x}$, as follows:

$$Z(\mathbf{x}) = \sum_{\mathbf{y}' \in \mathcal{Y}} \exp\Big(\sum_{i=1}^{|F_c|} w_i n_i(\mathbf{x}, \mathbf{y}')\Big) \tag{2.2}$$

2.2.2 MLN–EC: Probabilistic Event Calculus based on MLNs

MLN–EC [Skarlatidis, 2014; Skarlatidis et al., 2015, 2011] is a method which combines a discrete variant of the Event Calculus with the probabilistic framework of Markov Logic Networks in order to deal with uncertainty and retain the advantages of a logic-based representation. Figure 2.2 outlines the structure of the method. The input to MLN–EC is a stream of SDE occurrences, as well as a set of domain-dependent CE definitions. These definitions take the form of common-sense rules and describe the conditions under which a CE starts or ends. The basic concept behind MLN–EC is that it combines the given CE definitions with the domain-independent axioms of MLN–EC (2.3)–(2.6), generating a compact knowledge base serving as a pattern for the production of Markov Networks and enabling us to perform probabilistic inference and machine learning. The compact knowledge base is generated by performing circumscription by predicate completion [Lifschitz, 1994; Mueller, 2008; Shanahan, 1999] – a syntactic transformation where formulas are translated into logically stronger ones. The aim of circumscription is to automatically rule out all conditions which are not explicitly entailed by the given formulas. Hence, circumscription introduces a closed-world assumption to first-order logic.

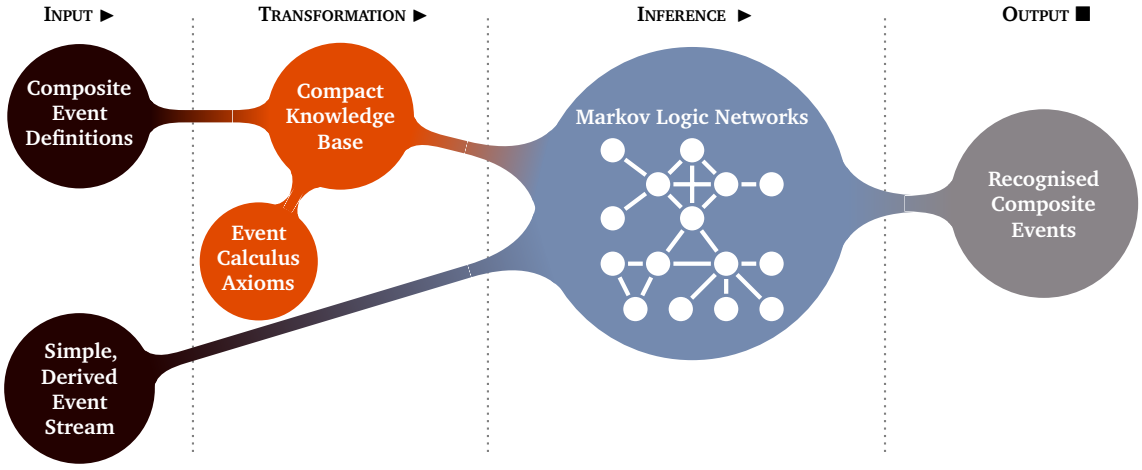

FIGURE 2.2: The structure of MLN–EC (after [Skarlatidis, 2014])

MLN–EC is based on a discrete version of the Event Calculus in first-order logic. Table 2.1 summarizes the main elements of MLN–EC. Following the conventions of first-order logic, variables and functions start with a lower-case letter and are assumed to be universally quantified unless otherwise indicated. Predicates and constants start with an upper-case letter.

The predicate `HappensAt` expresses the input evidence, determining the occurrence of a SDE at a specific time-point. The input stream of observed SDEs is represented as a narrative of ground `HappensAt` predicates. The predicates `InitiatedAt` and `TerminatedAt` specify the conditions under which a fluent – representing a CE – is to be initiated or terminated at a specific time-point.

Predicate	Meaning
HappensAt(e, t)	Event e occurs at time-point t
HoldsAt(f, t)	Fluent f holds at time-point t
InitiatedAt(f, t)	Fluent f is initiated at time-point t
TerminatedAt(f, t)	Fluent f is terminated at time-point t

TABLE 2.1: The MLN–EC predicates

The MLN–EC axioms that determine when a fluent holds are defined as follows:

$$\begin{aligned} &\texttt{HoldsAt}(f, t{+}1) \Leftarrow \\ &\qquad \texttt{InitiatedAt}(f, t) \end{aligned} \tag{2.3}$$

$$\begin{aligned} &\texttt{HoldsAt}(f, t{+}1) \Leftarrow \\ &\qquad \texttt{HoldsAt}(f, t) \wedge \\ &\qquad \neg\texttt{TerminatedAt}(f, t) \end{aligned} \tag{2.4}$$

Axiom (2.3) defines that if a fluent f is initiated at time t, then it holds at the next time-point. Axiom (2.4) specifies that a fluent continues to hold unless it is terminated.

The axioms that determine when a fluent does not hold are defined similarly:

$$\begin{aligned} &\neg\texttt{HoldsAt}(f, t{+}1) \Leftarrow \\ &\qquad \texttt{TerminatedAt}(f, t) \end{aligned} \tag{2.5}$$

$$\begin{aligned} &\neg\texttt{HoldsAt}(f, t{+}1) \Leftarrow \\ &\qquad \neg\texttt{HoldsAt}(f, t) \wedge \\ &\qquad \neg\texttt{InitiatedAt}(f, t) \end{aligned} \tag{2.6}$$

According to Axiom (2.5), if a fluent f is terminated at time t then it does not hold at the next time-point. Axiom (2.6) states that a fluent continues not to hold unless it is initiated.

2.2.3 Probabilistic Inference

In order to be able to perform machine learning using MLNs, efficient methods for probabilistic inference are required. Directly computing the Equation (2.1) is intractable, because the value of $Z(\mathbf{x})$, shown in Equation (2.2), depends on the relationship among all clauses in the knowledge base. For this reason, a variety of efficient inference algorithms have been proposed, based on local search and sampling [Biba et al., 2011; Poon and Domingos, 2006; Singla and Domingos, 2006], variants of Belief Propagation [Gonzalez et al., 2009; Kersting, 2012; Kersting et al., 2009; Singla and Domingos, 2008], Integer Linear Programming [Huynh and Mooney, 2009; Noessner et al., 2013; Riedel, 2008], Lifted Model Counting [Apsel and Brafman, 2012; den Broeck et al., 2011;

Gogate and Domingos, 2011], etc. Below we present the two types of inference that can be performed in MLNs: marginal inference and maximum a-posteriori inference (MAP).

2.2.3.1 Marginal Inference

Marginal inference computes the conditional probability that CEs hold given an input of observed SDEs:

$$P(\texttt{HoldsAt}(\texttt{CE}, \texttt{T}) = True \mid SDEs)$$

The probability value measures the confidence that the CE is recognized. Since it is #P-complete to compute it [Domingos and Lowd, 2009], approximate Markov Chain Monte Carlo (MCMC) sampling algorithms have been employed for efficient inference.

Since MLNs incorporate logic to probabilistic modeling, inference must deal both with deterministic and probabilistic dependencies. Deterministic or near-deterministic dependencies arise from formulas accompanied by an infinite or a strong weight value. On the other hand, by being purely statistical, MCMC can only handle probabilistic dependencies, because in the presence of deterministic dependencies the properties of *ergodicity* and *detailed balance* characterizing Markov Chains are violated and the sampling algorithms give poor results [Poon and Domingos, 2006]. Ergodicity requires that all states are reachable from each other, while detailed balance that the probability of moving from one state to another is identical to the probability of moving back. Deterministic dependencies create isolated regions in the state space and thus typical MCMC methods, such as Gibbs sampling [Casella and George, 1992], get trapped in local regions and are unable to converge.

To overcome these issues, MC-SAT algorithm was developed [Poon and Domingos, 2006], which is a MCMC method that combines satisfiability testing with *slice-sampling* [Damlen et al., 1999]. Specifically, a satisfiability solver is used to find an assignment satisfying all hard-constraint clauses and subsequently a sequence of sampling steps takes place, each one choosing a subset of ground clauses satisfied by the previous step in order to be satisfied in the next step. The sampling is restricted only to states that satisfy at least all chosen clauses. Therefore, MCMC cannot get trapped in local regions, because satisfiability testing collects samples from all isolated regions.

2.2.3.2 MAP Inference

MAP inference, on the other hand, identifies the most probable assignment among all `HoldsAt` instantiations that are consistent with the given input of observed SDEs:

$$\underset{\texttt{HoldsAt}}{\operatorname{argmax}}\, P(\texttt{HoldsAt(CE, T)} \mid \mathit{SDEs})$$

This task reduces to finding the truth assignment of all `HoldsAt` instantiations that maximizes the sum of weights of satisfied ground clauses. This is equivalent to the weighted maximum satisfiability problem. The problem is NP-hard in general and there has been significant work on finding an approximate solution efficiently using local search algorithms (e.g., MaxWalkSAT [Kautz et al., 1997], see Hoos and Stützle [2004] for in depth analysis) and linear programming methods (e.g., Huynh and Mooney [2009]; Noessner et al. [2013]; Riedel [2008]).

We employ both approaches for approximate MAP inference by using the classic local search MaxWalkSAT solver proposed by Kautz et al. [1997] and the LP-relaxed Integer Linear Programming method proposed by Huynh and Mooney [2009]. In particular, for the latter algorithm, the ground Markov network is translated into a set of linear constraints, as shown in Figure 2.3, and solved using standard linear optimization algorithms. Note that a binary variable y_i is assigned to each unknown ground predicate. For each (non-unit[2]) soft-constrained clause, an auxiliary binary variable z_j is introduced and additional constraints are added containing the binary variables of the literals appearing in the clause (negated literals are appearing as $1-y_i$). Negative weighted clauses impose a constraint for each individual literal in the clause indicating that it should not be satisfied. Hard-constrained clauses impose constraints causing them to necessarily be satisfied. Unit clauses do not impose any constraints.

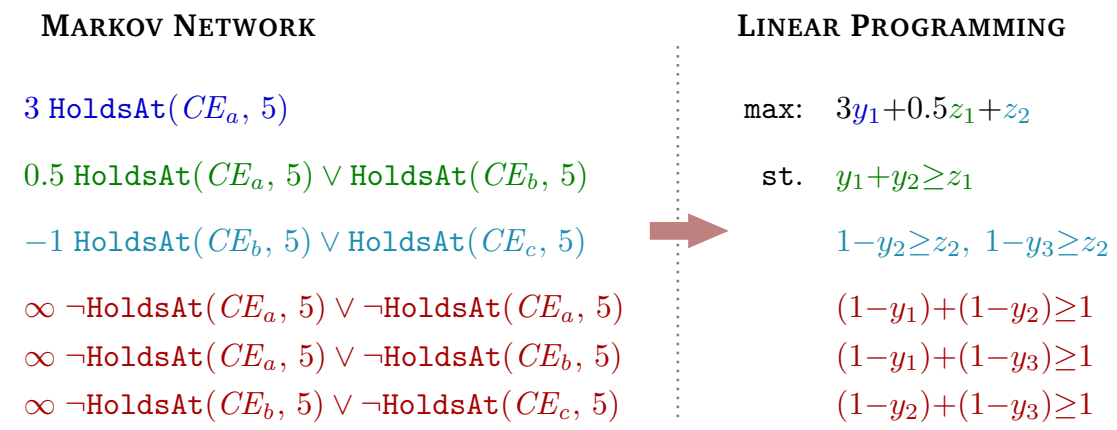

FIGURE 2.3: Markov network translation into an equivalent optimization problem

Due to the NP-hardness of the problem, the linear programming solver usually returns non-integral solutions — i.e., the assignment of some ground $\texttt{HoldsAt(CE, T)}$ is not Boolean, but within the open interval $(0, 1)$. For that reason, the method uses a rounding procedure called ROUNDUP [Boros and Hammer, 2002]. Specifically, the procedure

[2]Unit clauses are composed of a single literal.

iteratively assigns the truth value of non-integral ground atoms, by satisfying the clauses that appear with respect to their cost (i.e., summation of their weights). Compared to the local search MaxWalkSAT algorithm, the linear programming approach typically achieves higher accuracy [Huynh and Mooney, 2009].

2.2.4 Weight Learning

The weights of the soft-constrained clauses in MLNs can be estimated from training data, using supervised learning techniques. When the goal is to learn a model that recognizes CEs with some confidence (i.e., probability), then a widely adopted learning approach is to minimize the negative conditional log-likelihood (CLL) function that is derived from Equation (2.1). Given training data that are composed of a set X of evidence predicates (e.g., ground `HappensAt` predicates) and their corresponding query predicates Y (e.g., ground `HoldsAt` predicates), the negative CLL has the following form:

$$-\log P_w(Y{=}\mathbf{y} \mid X{=}\mathbf{x}) = \log Z(\mathbf{x}) - \sum_{i=1}^{|F_c|} w_i n_i(\mathbf{x}, \mathbf{y})$$

The vector $\mathbf{x}$ is the assignment of truth values to the evidence random variables X, according to the training data. Similarly, $\mathbf{y}$ represents a possible assignment of truth values to the query random variables Y that are provided as annotation. CLL is used to evaluate how well the model fits the given training data.

The parameters of the model are the weight values w_i that are associated to the soft-constrained clauses in F_c and can be estimated by using either *first-order* or *second-order* optimization methods [Lowd and Domingos, 2007; Singla and Domingos, 2005]. First-order methods apply standard gradient descent optimization techniques, e.g., the Voted Perceptron algorithm [Collins, 2002; Singla and Domingos, 2005], while second-order methods pick a search direction based on the quadratic approximation of the target function. As stated by Lowd and Domingos [2007], second-order methods are more appropriate for MLN training, as they do not suffer from the problem of *ill-conditioning*. In a training set some clauses may have a significantly greater number of satisfied groundings than others, causing the variance of their counts to be correspondingly larger. This situation makes the convergence of the standard gradient descent methods very slow, since there is no single appropriate learning rate for all soft-constrained clauses.

If the goal is to predict accurate target-predicate probabilities (e.g., the probability of a CE to hold), these approaches are well motivated. However, in many applications, the actual goal is to maximize an alternative performance metric such as classification accuracy or F-measure. Thus, an alternative to CLL optimization is max-margin training which constitutes an approach competitive to discriminative training and also has the advantage that it can be adapted to maximize a variety of performance metrics beyond

classification accuracy such as ROC area, Precision or Recall [Joachims, 2005]. Furthermore, max-margin methods have been successfully applied to structure prediction, where the output space consists of structured objects, such as sequences, trees or graphs [Taskar et al., 2003; Tsochantaridis et al., 2005] in order to learn parameters that maximize the margin between the probability of the correct assignment and that of the closest incorrect assignment. Instead of optimizing CLL, max-margin maximize the following ratio:

$$\frac{P(Y{=}\mathbf{y}|X{=}\mathbf{x},\mathbf{w})}{P(Y{=}\hat{\mathbf{y}}|X{=}\mathbf{x},\mathbf{w})}$$

The above equation measures the ratio between the probability of correct truth assignment $\mathbf{y}$ of CEs and the closest competing incorrect truth assignment, which is also known as separation oracle, $\hat{\mathbf{y}}{=}\text{argmax}_{\bar{\mathbf{y}}\in\mathbf{Y}\setminus\mathbf{y}}P(Y{=}\bar{\mathbf{y}}|X{=}\mathbf{x})$. The intuition is that the more accurate is a model the closer should be in the correct truth assignment and thus the ratio should be minimized. We employ the method of Huynh and Mooney [2009] which formulates the max-margin problem as 1-slack structural support vector machine (SVM) using a cutting-plane algorithm proposed by Joachims et al. [2009]. Specifically, structural SVMs are predicting structured outputs instead of simple labels or real values. They describe the problem of learning a function $h : \mathcal{X} \mapsto \mathcal{Y}$, where $\mathcal{X}$ is the space of inputs examples, and $\mathcal{Y}$ is the space of multivariate and structured outputs from the set of training examples $S = ((\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)) \in (\mathcal{X} \times \mathcal{Y})^n$.

The goal is to find a function h that has low prediction error. This can be accomplished by learning a discriminant function $f : \mathcal{X} \times \mathcal{Y} \mapsto \mathcal{R}$ and maximizing f over all $\mathbf{y} \in \mathcal{Y}$ for a given input $\mathbf{x}$ to get a classifier of the form:

$$h_w(\mathbf{x}) = \underset{\mathbf{y}\in\mathcal{Y}}{\text{argmax}} f_w(\mathbf{x}, \mathbf{y})$$

The discriminant function $f_w(\mathbf{x}, \mathbf{y}) = \mathbf{w}^T\mathbf{\Psi}(\mathbf{x}, \mathbf{y})$ is linear in the space of features, where $\mathbf{w} \in \mathcal{R}^N$ is a parameter vector and $\mathbf{\Psi}(\mathbf{x}, \mathbf{y})$ is a feature vector relating an input $\mathbf{x}$ and output $\mathbf{y}$. The features need to be designed for a given problem so that they capture the dependency structure of $\mathbf{y}$ and $\mathbf{x}$ and the relations among the outputs $\mathbf{y}$. In our case $\mathbf{\Psi}(\mathbf{x}, \mathbf{y}) = \mathbf{n}(\mathbf{x}, \mathbf{y})$ which is the number of satisfied groundings in $\mathbf{x}$ and $\mathbf{y}$. Therefore, the goal is to find a parameter vector $\mathbf{w}$ that maximises the margin by employing linear programming techniques, such as the Integer Linear Programming inference method described above.

2.2.4.1 Online Weight Learning

Batch training algorithms, described in the previous section, must repeatedly run inference over all training examples in each iteration. This becomes computationally

expensive and eventually infeasible for very large datasets with thousands of training examples, which may not even fit in the main memory. To overcome this problem, we employ an online learning strategy where the learner sequentially processes a set of examples at each step in the form of micro-batches. The size of these batches can be adjusted.

In particular, we employ an online max-margin method proposed by Huynh and Mooney [2011a] based on the Coordinate-Dual-Ascent update rule (CDA). The algorithm is derived from the primal-dual framework for strongly convex loss functions [Hazan et al., 2006], which is a framework for deriving online algorithms that have low regret. CDA can use various loss functions to guide the optimization. We employ a prediction-based loss which measures the difference between the predicted possible world and the ground-truth one. The update rule in terms of the weight vector for each step t is the following:

$$\mathbf{w}_{t+1} = \frac{t-1}{t}\mathbf{w}_t + \min\Big\{\frac{1}{\sigma t}, \frac{\ell_t}{||\Delta\phi_t||_2^2}\Big\}\Delta\phi_t \tag{2.7}$$

where σ is a non-negative constant and ϕ_t is a feature vector representing the number of satisfied groundings in $\mathbf{x}$ and $\mathbf{y}$ as in the batch version of max-margin training. Note that the learning rate of the update rule is controlled by the loss ℓ_t suffered at each step. At the beginning, when the model has low predictive quality, CDA aggressively updates the model based on the loss suffered at each step. Later, when the model improves, it is updated less aggressively.

Although CDA is quite fast, it may lack in accuracy. Moreover, Lee et al. [2006] states that parameter learning algorithms used by the structure learning procedure, which may introduce a lot of new features and many of which may not be useful, perform better when they use L1-regularization. Therefore, L1-regularization, which has the tendency to force parameters to zero and thus leads us to sparser models, is preferred over other Lp norm regularization methods. For this reason we also employ another algorithm for online optimization used by Huynh and Mooney [2011b], in order to perform structure learning more efficiently.

AdaGrad, proposed by Duchi et al. [2011], is a state-of-the-art online method, which is an L1-regularized adaptive subgradient algorithm based on composite mirror-descent updates. AdaGrad belongs to a family of subgradient methods that dynamically incorporate knowledge of the geometry of the data observed in earlier steps to perform more informative gradient-based learning. Informally, the algorithm gives frequently occurring features very low learning rates and infrequent features high learning rates, where the intuition is that each time an infrequent feature is seen, the learner should “take notice". Thus, the adaptation facilitates finding and identifying very predictive but comparatively rare features. It is often the case that infrequently occurring features are highly informative and discriminative. Indeed, in many applications of online and stochastic learning,

the input instances are of very high dimensionality, yet within any particular instance only a few features are non-zero. AdaGrad updates the weight vector at each step t as follows:

$$\mathbf{w}_{t+1,i} = \text{sign}\Big(\mathbf{w}_{t,i} - \frac{\eta}{H_{t,ii}}\mathbf{g}_{t,i}\Big)\Big[\Big|\mathbf{w}_{t,i} - \frac{\eta}{H_{t,ii}}\mathbf{g}_{t,i}\Big| - \frac{\lambda\eta}{H_{t,ii}}\Big]_+ \tag{2.8}$$

where λ is the regularization parameter used to tackle overfitting, η is the learning rate, $\mathbf{w}_{t,i}$ is the i-th component of the weight vector at step t and $[a]_+$ denotes a truncated function at 0, i.e. $[a]_+ = \max(a, 0)$. The function sign returns the sign of the operation inside the brackets. The subgradient $\mathbf{g}_t$ is computed, at each step t, over the loss function which guides the optimization process, as follows:

$$\mathbf{g}_{PL} = \mathbf{n}_C(\mathbf{x}_t, \mathbf{y}_t^{PL}) - \mathbf{n}_C(\mathbf{x}_t, \mathbf{y}_t) = \Delta\mathbf{n}_C \tag{2.9}$$

where $\mathbf{n}_C$ is the number of true groundings for the clause C and $\mathbf{y}_t^{PL}$ is the predicted possible world. The subgradient of the loss function represents the difference between the number of satisfied groundings computed over the predicted possible world and the ground-truth one. Regarding the loss function, we use the prediction-based loss function, also used by Huynh and Mooney [2011b] which is a simpler variant of the max-margin loss [Huynh and Mooney, 2009]. Furthermore, the subgradient is also required to compute $H_{t,ii}$ as follows,

$$H_{t,ii} = \delta + ||\mathbf{g}_{1:t,i}||_2 = \delta + \sqrt{\sum_{j=1}^{t}\mathbf{g}_{j,i}^2} \tag{2.10}$$

where $\mathbf{H}$ is a diagonal matrix and δ is the default value of the matrix. Note that AdaGrad assigns a different step size $\frac{\eta}{H_{t,ii}}$ for each component of the weight vector. Thus, AdaGrad needs to retain the sum of squared subgradients of each component. From the update rule we can see that if a clause is not relevant to the current example the AdaGrad discounts its weight by $\frac{\lambda\eta}{H_{t,ii}}$. Thus, irrelevant clauses will be zeroed out in the long run.

2.2.5 LoMRF: Logical Markov Random Fields

There are several systems implementing Markov Logic Networks reasoning and learning algorithms such as, Alchemy, Tuffy, Markov TheBeast, ProbCob, RockIt, Factorie and LoMRF. LoMRF[3] [Skarlatidis, 2012] is an open-source implementation of Markov Logic Networks written in Scala programming language. It is a tool for easily performing knowledge base compilation, grounding, exporting the Markov network in various

[3] https://github.com/anskarl/LoMRF

formats as well as performing MAP and marginal inference. It provides various useful features such as predicate completion in three variants (full, decomposed, and simplification) [Lifschitz, 1994; Mueller, 2008; Shanahan, 1999], clausal form transformation, function elimination and introduction by using auxiliary predicates. Furthermore, it also provides a parallel grounding algorithm which constructs the minimal Markov Network or Markov Random Field (MRF) required by doing various optimizations in the process (removing tautologies, ignoring everything not required for inferring the query variables, etc.). This implementation, in particular, was very suitable to our work due to the fact that during event recognition we usually query for every time point and therefore even lifted inference may not be a solution. For all these reasons we chose LoMRF as a baseline for this work and contributed to the project various features that extended its functionality. Specifically, we contributed another algorithm for MAP inference based on integer linear programming, as well as algorithms for parameter learning (max-margin, CDA, and AdaGrad) and online structure learning (OSL and OSLα).

2.3 Related Work

The task of structure learning is to discover the dependency structure of the model as well as estimate the parameters of these dependencies given a set of training examples $\boldsymbol{\mathcal{D}}$. The training set usually consists of a single interconnected example containing many ground instances of observed SDEs as well as unobserved CEs (query predicates). Nearly all approaches developed for structure learning perform a heuristic search through the space of possible structures, known as the hypothesis space $\boldsymbol{\mathcal{H}}$, in order to avoid exhaustive search, which is combinatorially explosive for complex models. Typically, given a particular scoring function $\mathcal{S}(h, \boldsymbol{\mathcal{D}})$ for $h \in \boldsymbol{\mathcal{H}}$ the task is reduced to finding a hypothesis $h^* \in \boldsymbol{\mathcal{H}}$ that maximizes the score, i.e., $h^* = \operatorname{argmax}_{h \in \boldsymbol{\mathcal{H}}} \mathcal{S}(h, \boldsymbol{\mathcal{D}})$.

In MLNs the structure is a set of weighted formulas in conjunctive normal form. In principle, the structure can be learned or revised by using approaches for learning graphical models [Heckerman, 1999; McCallum, 2012; Pietra et al., 1997], as well as Inductive Logic Programming (ILP) techniques [De Raedt and Dehaspe, 1997; Quinlan, 1990; Srinivasan, 2004]. However, since an MLN represents a probability distribution over possible worlds, much better results are obtained by using evaluation functions based on likelihood (e.g., pseudo-likelihood), rather than typical ILP ones like accuracy and coverage [Kok and Domingos, 2005]. Log-likelihood or conditional log-likehood are probably the best evaluation functions, albeit particularly expensive to compute.

In this section we outline important batch approaches that have been developed in order to efficiently learn and revise the MLN structure, starting either from an existing knowledge base or an empty one. These methods have proven to be beneficial in many real-world applications in citation analysis, web mining, natural language processing, robotics, bioinformatics, computer games as well as activity recognition.

Top-down structure learning Top-down structure learning as proposed by Kok and Domingos [2005] learns and/or revises the MLN structure in an iterative fashion. The initial structure can be an empty network or an existing KB. At each step, the algorithm searches for the best clause to add to the model. Searching can be performed using one of two possible strategies. The first one, *beam search*, keeps the best k clause candidates at each step of the search. The second one, *shortest-first search*, tries to find the best clauses of length i before it moves on to clauses of length $i + 1$. Candidate clauses are formed by adding each predicate (negated or not) to each current clause, using all possible combinations of variables, subject to the constraint that at least one variable in the new predicate must appear in the current clause. Candidate clauses are scored using the weighted pseudo log-likelihood measure, an adaptation of the pseudo log-likelihood that weights the pseudo-likelihood of each grounded atom by 1 over the number of groundings of its predicate, in order to prevent predicates with larger arity from dominating the expression. The procedure follows a blind generate-and-test strategy in which many potential changes to an existing model are systematically generated independently of the training data, and then tested for empirical adequacy. For complex models such as MLNs, the space of potential revisions is combinatorially explosive and such a search procedure can become difficult to control, resulting in convergence to suboptimal local maxima.

Iterative Local Search One way of addressing the potential shortcomings of greedy structure selection is by using iterative local search techniques [Lourenço et al., 2003], that explores the space of structures through a biased sampling of the set of local optima found by a local search procedure. They focus the search not on the full space of solutions but on a smaller subspace defined by the solutions that are locally optimal for the optimization engine. These techniques alternate between two types of search steps: (a) either moving towards a locally optimal solution using a given evaluation function, or (b) perturbing the current solution in order to escape from local optima. The algorithm proposed by Biba et al. [2008] uses the above technique to avoid local maxima when learning MLNs in a discriminative setting, where the focus is on predicting a specific target predicate given the evidence on all other predicates.

Bottom-up structure learning Another alternative is using algorithms developed to restrict the hypothesis space $\mathcal{H}$, typically by performing a pre-processing step that, roughly speaking, discovers more promising regions of the space. Bottom-Up Structure Learning (BUSL) introduced by Mihalkova and Mooney [2007] is based on the observation that, once an MLN is instantiated into a Markov network (through the grounding procedure), the instantiations of each clause of the MLN define a set of identically structured cliques in the Markov network. BUSL inverts this process of instantiation and constrains the search space by inducing lifted templates for such cliques in order to learn a so-called *Markov network template*. The template network is composed of *template nodes*, conjunctions of one or more literals that serve as building blocks for creating clauses. Template nodes are

constructed by looking for groups of constant-sharing ground literals that are true in the data and abstracting them by substituting variables for constants. Thus, these template nodes could also be viewed as portions of clauses that have true groundings in the data. To understand why conjunctions of literals with true groundings are good candidates for clause components, consider the special case of a definite clause $L_1 \wedge \cdots \wedge L_n \Rightarrow P$. If the conjoined literals in the body have no true groundings, then the clause is always trivially satisfied. Therefore, true conjunctions will be most useful for building effective clauses. To search for these, BUSL generates clause candidates by focusing on each maximal clique in turn and producing all possible clauses consistent with it. The candidates are then evaluated using the weighted pseudo log-likelihood score. BUSL restricts its search for clauses only to those candidates whose literals correspond to template nodes that form a clique in the template. It also makes a number of additional restrictions on the search in order to decrease the number of free variables in a clause, thus decreasing the size of the ground MLN during inference, and further reducing the search space. Therefore, BUSL typically restricts the search to very short paths, creating short clauses from them and greedily joining them into longer ones. Although this approach is faster and yields more accurate models than the top-down approach, it may still converge to a suboptimal local maxima.

Discriminative heuristic structure learning An approach somewhat similar to the principle underlying the BUSL algorithm was proposed by Dinh et al. [2010]. Heuristic Method for Discriminative Structure Learning employs a heuristic method in order to discriminatively learn the structure of MLNs. It consists of three main steps. First, for each true ground query atom, applies a heuristic technique to build a set of variable literals, called chain. To achieve that it adds to the set each ground atom in the training example $\mathcal{D}$ connected to the ground query atom through its arguments and variabilize it. Then transforms the learning dataset to a boolean table, having ground query atoms as rows and variable literals as columns, representing their connections. Second, by applying the Grow-Shrink Markov Network [Bromberg F, 2009] algorithm to these boolean tables, it extracts a set of template clauses. A template clause is a disjunction of positive variable literals. Finally, candidate clauses are built from the template clauses to be added into the MLN. The score of the weighted MLN is then measured by computing either its conditional log-likelihood or weighted pseudo log-likelihood. The procedure follows a similar approach to the BUSL algorithm. Both of them consist of three mains steps: Transforming the relational dataset into template clauses, building candidate clauses using these templates and putting clauses into the MLN. Although the quality of the boolean tables constructed yields higher accuracy, the greedy strategy delivers higher running times.

Moralized Bayes Nets An alternative approach is searching for structures of increasing complexity at each stage of the procedure using the structures found at the previous stage

to constrain the search space. Such a strategy was employed by Khosravi et al. [2010] for learning MLN structure in domains that contain many descriptive attributes, namely predicates having very large arity. Their approach, which is similar to the technique employed to constrain the search space in Probabilistic Relational Models [Friedman et al., 1999], distinguishes between two types of tables – attribute tables that describe a single entity type, and relationship tables that describe relationships between entities. The algorithm, called MBN (Moralized Bayes Net), proceeds in three stages. In the first stage, dependencies local to attribute tables are learned. In the second stage, dependencies over a join of an attribute table and a relationship table are learned, but the search space is constrained by requiring that all dependencies local to the attribute table found in the first stage remain the same. Finally, in the third stage, dependencies over a join of two relationship tables, joined with relevant attribute tables are learned and the search space is similarly constrained. Although the goal of MBN is to learn an undirected model, dependencies are learned using a Bayesian network learner and then the directed structures are converted to undirected ones by using moralization [Cowell et al., 2007]. The advantage of this approach is that structure learning in directed models is significantly faster than structure learning in undirected models due to the decomposability of the score, which allows it to be updated locally, only in parts of the structure that have been modified, making scoring of candidate structures more efficient. On the other hand, when scaling to larger table joins, the algorithm becomes computationally intensive.

Hypergraph Lifting Another MLN learner that is based on constraining the search space is the Learning via Hypergraph Lifting (LHL) algorithm introduced by Kok and Domingos [2009]. LHL receives as input the set of clause candidates considered by relational pathfinding [Richards and Mooney, 1992] and focuses only on the most promising ones. Developed in the ILP community, relational pathfinding searches for clauses by tracing paths across the true instantiations of relations in the data. The data are represented as a generalized graph, called hypergraph, having edges connecting any number of nodes, called hyperedges. However, because in real-world relational domains the search space over relational paths may be very large, a crucial aspect of LHL is that it does not perform relational pathfinding over the original relational graph of the data, but over a so-called lifted hypergraph, which is formed by jointly clustering the entities in the domain via an agglomerative clustering procedure. Intuitively, constants are clustered together if they tend to participate in the same kind of relations (i.e., predicates) with constants from other clusters. The complexity of the agglomerative clustering in the general case is $O(n^3)$, which makes it slow for large datasets. On the other hand, pathfinding on this lifted hypergraph is typically at least an order of magnitude faster than on the ground training data, producing MLNs that are more accurate than previous approaches. Subsequently, the ground atoms in each path are variabilized, and they are used to form clauses, which are evaluated using a pseudo-likelihood measure. Then, the

algorithm iterates over the clauses from shortest to longest and for each clause, compares its score against those of its sub-clauses. If a clause scores higher than all the sub-clauses it is retained, otherwise it is discarded because it is unlikely to be useful. Finally, the retained clauses are added to an MLN, the weights are re-learnt and the clauses are kept in the MLN, which improves the overall weighted pseudo log-likelihood. LHL is a data-driven algorithm which cannot exploit background knowledge for learning rules. This inability makes it inappropriate for capturing complex relations and learning qualitatively meaningful rules.

Structural Motifs Kok and Domingos [2010] have proposed constraining the search for clauses by identifying so-called *structural motifs*, which capture commonly occurring patterns among densely connected entities in the domain (i.e. sets of entities that are closely related). The Learning using Structural Motifs (LSM) algorithm, a modification of LHL, proceeds by first identifying motifs (recurring patterns) and then searching for clauses by performing relational pathfinding within them. To discover motifs, LSM starts from an entity i in the relational graph and performs a series of random walks. The random walks are used to calculate the average number of steps required to reach another entity j for the first time, called hitting time. Entities that are reachable within a thresholded hitting time as well as the hyperedges among them are included in the motif and the paths through which they are reachable from i are recorded. Next, the entities included in the motif are clustered by their hitting times into groups of potentially symmetrical nodes (i.e., nodes that have symmetrical paths). The nodes within each group are then further clustered in an agglomerative manner by the similarity of distributions over paths through which they are reachable from i. This process results in a lifted hypergraph, analogous to the one produced by LHL; however, whereas in LHL nodes were clustered based on their close neighborhood in the relational graph, here they are clustered based on their longer-range connections to other nodes. Thus, intuitively, LHL is making use of length-2 paths to determine the similarity of nodes. In contrast, LSM uses longer paths, and thus more information, to find clusterings of nodes (motifs). In addition, LSM finds various clusterings rather than just a single one. Motifs are extracted from the lifted hypergraphs through depth-first search. Finally, LSM runs relational pathfinding on each motif to find candidate rules, and retains the good ones in an MLN using the same procedure as LHL. Although LSM is much faster and accurate than LHL, especially for more compex models, is still data-driven and thus cannot exploit background knowledge for learning rules.

Gradient-Based Boosting Khot et al. [2011] have extended the functional gradient boosting approach to learning the relational dependency networks of Natarajan et al. [2012] to MLNs. In contrast to previous approaches, they learn structure and parameters simultaneously, thus avoiding the cost of repeated parameter estimation. This is done through a sequence of functional gradient steps, each of which adds clauses based on the

point-wise gradients of the training examples in the current model. They present two representations for functional gradients. The first one is based on relational regression trees [Blockeel and De Raedt, 1998] and the second one learns Horn clauses by using a beam search that adds literals to clauses that reduce the squared error. Moreover, Khot et al. [2015] extend the algorithm to handle missing data by using an EM-based approach.

Markov Networks Structure learning techniques for Markov networks have been developed in the graphical models community. One technique is that of structure selection through appropriate regularization [Lee et al., 2006]. In this approach, a large number of factors of a Markov network are evaluated simultaneously by training parameters over them and using the L1 norm as a regularizer (as opposed to the typically used L2 norm). Since the L1 norm imposes a strong penalty on smaller parameters, its effect is that it forces more parameters to zero, leading to sparser models. Huynh and Mooney [2008] extend this technique for structure learning of MLNs by first using Aleph [Srinivasan, 2004], an off-the-shelf ILP learner, to generate a large set of potential factors (in this case, first-order clauses), and then perform L1-regularized parameter learning over this set. Another technique proposed by Lowd and Davis [2010] uses probabilistic decision trees to learn the structure. The Decision Tree Structure Learning (DTSL) algorithm learns probabilistic decision trees using the training data in a depth-first manner to predict the value of each variable. It then converts the trees into sets of conjunctive features. All learned features are merged into a global model and the parameters for those features can be estimated using any standard parameter learning method. In addition to this conversion, various pruning methods are used during the feature generation process in order to make learning and inference faster. Lowd and Davis [2014] extended the algorithm and introduced two new variations of DTSL. The first one, DT-BLM, builds on DTSL by using the Bottom-Up Learning of Markov Networks (BLM) algorithm of Davis and Domingos [2010] to further refine the structure learned by DTSL. This algorithm is much slower, but usually more accurate than DTSL. Furthermore, it serves as an example of how decision trees can be used to improve search-based structure learning algorithms, by providing a good initial structure. The second one, DT+L1, combines the structure learned by DTSL with the pairwise interactions learned by L1-regularized logistic regression [Ravikumar et al., 2010] by taking the union of the best DTSL and L1 feature set. The trees used by DTSL are good at capturing higher-order interactions. In contrast, L1 captures many independent interaction terms, but each interaction can only be between just two variables. Their combination offers the potential to represent both kinds of interaction, leading to better performance in many domains.

Online Structure Learning The Online Structure Learning (OSL) algorithm proposed by Huynh and Mooney [2011b] updates both the structure and the parameters of the model using an incremental approach based on the model's incorrect predictions. Unlike

the previously presented methods, OSL takes into account the predicted possible worlds, i.e., the most probable possible worlds predicted by the current model. Specifically, at each step t of the algorithm, if the predicted possible world y_t^P is different from the ground-truth one y_t then OSL focuses on searching for clauses that differentiate y_t from y_t^P. This is related to the idea of using implicit negative examples in ILP [Zelle et al., 1995]. In this particular case, each ground-truth possible world plays the role of a positive example and any predicted possible world that differs from the ground-truth one is incorrect and can be considered as a negative example. In addition, this follows the max-margin training criterion which focuses on discriminating the true label from the most probable incorrect one [Tsochantaridis et al., 2005]. In order to discover useful clauses specific to the set of wrongly predicted atoms, OSL employs relational pathfinding over a hypergraph [Richards and Mooney, 1992]. The search procedure is also combined with mode declarations [Muggleton, 1995], a form of language bias, to speed up the process. Paths found by the mode-guided relational pathfinding process are generalized into first-order clauses by replacing constants with variables. The resulting set of clauses is added to an MLN and the parameters are updated using the AdaGrad weight learning algorithm described in Section 2.2.4.

2.4 Summary

The aforementioned methods for structure learning in Markov Logic Networks, excluding OSL, are batch learning algorithms that are effectively designed for training data consisting of many ground instances. Moreover, most of these algorithms are strictly data-driven, which means that they only consider the ground-truth possible worlds and search for clauses that improve the likelihood of those worlds. They do not exploit the background knowledge that may be available about a task and may spend a lot of time exploring clauses that are true in most of these worlds, therefore largely useless for the purposes of learning. To overcome these issues we developed `OSL`α, presented in Chapter 3, that exploits the background knowledge domain-independent axiomatization in order constrain the seach space of possible structures. Moreover, it uses an online strategy, like OSL, in order to effectively handle large datasets.

3 | OSLα: Online Structure Learning using background knowledge Axiomatization

"Reasoning draws a conclusion, but does not make the conclusion certain, unless the mind discovers it by the path of experience."
— Roger Bacon

As mentioned in Section 1.1, our goal is to effectively learn definitions from a search space having a very large domain of constants by exploiting the background knowledge axiomatization of a specific task. In our case we take advantage of the domain-independent rules of MLN–EC in order to learn domain specific composite event (CE) definitions and accurately perform event recognition given an input of observed simple derived events (SDEs). OSL has several limitations which renders it unable to learn such definitions. Specifically, even when performing mode-guided search over the hypergraph, the space of possible paths can become exponentially large. For instance, the Event Calculus is a temporal formalism describing CE occurrences over time. Therefore, data used for training will inevitably contain a large domain of time points (possibly) having multiple complex temporal relations between events. Mode declarations alone cannot handle this large domain. It will be then fundamental to prune a portion of the search space. Moreover, existing structure learning methods, including OSL, assume that domains do not contain any functions, which are essential for the Event Calculus representation.

To cope with these limitations we propose OSLα, which exploits domain-independent axioms of the background knowledge, to further constrain the search space only to clauses subject to specific characteristics introduced by these axioms. Furthermore, our approach can handle a subset of functions defined by the first-order logic formalism and effectively learn definitions using these functions. Figure 3.1 presents the interactions among the components underlying OSLα. In our case, the background knowledge consists of the MLN–EC axioms (i.e., domain-independent rules) and an already known hypothesis (i.e., set of clauses). At a step t_n of the online procedure a training example (micro-batch) $\mathcal{D}_{t_n}$ arrives and is used together with the already learnt hypothesis in order to

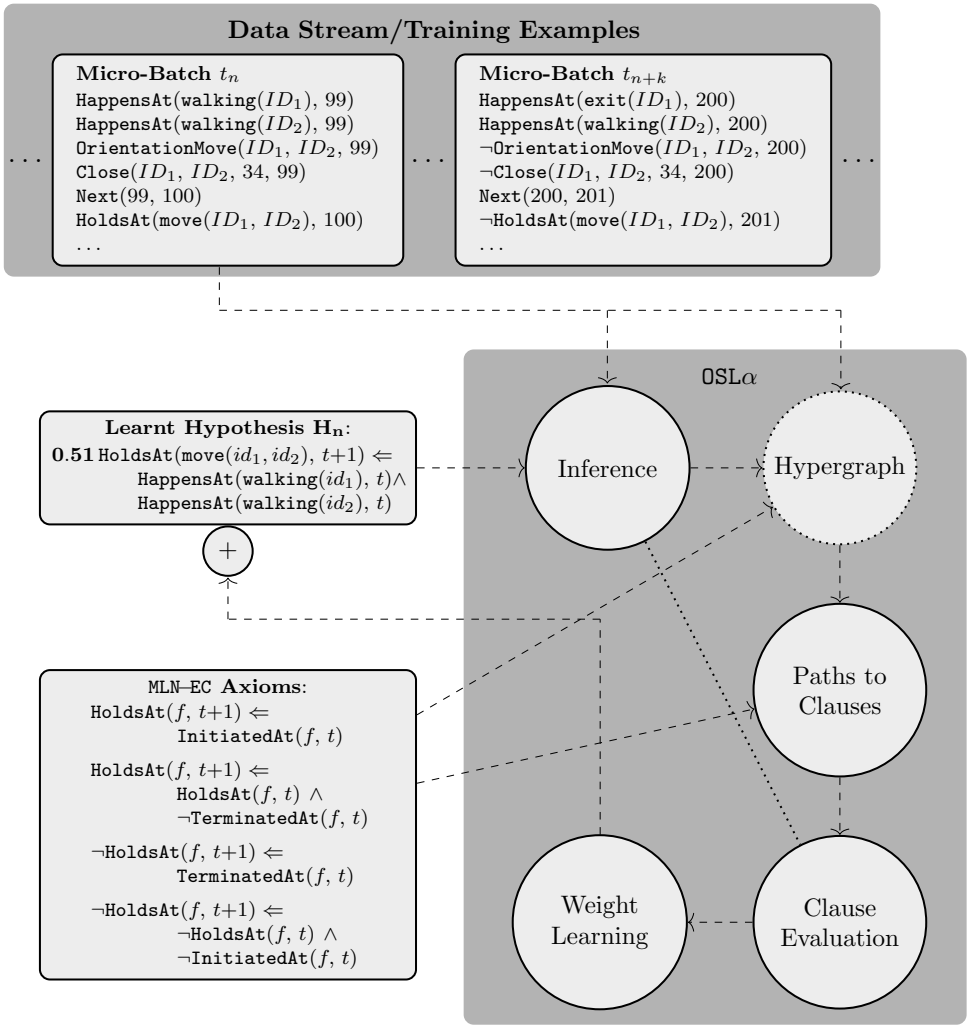

FIGURE 3.1: The procedure of OSLα

predict the truth values $y^P_{t_n}$ of the CEs of interest using probabilistic inference. Then for all wrongly predicted CEs the hypergraph is searched, guided by MLN–EC axioms, for definite clauses explaining these CEs. The paths discovered during the search are then translated into clauses and evaluated. The resulting set of retained clauses is passed onto the weight learning module for estimating their weights. Then, the set of weighted clauses is appended into the hypothesis $\boldsymbol{\mathcal{H}}_n$ and the whole procedure is repeated given the next training example $\boldsymbol{\mathcal{D}}_{t_{n+1}}$.

In Section 3.1 we describe the procedure of grouping the axioms into templates used by the hypergraph to constrain the space of possible structures. In Section 3.2 we present the hypergraph search by relational pathfinding and mode declarations alone and in Section 3.3 the extended template-guided search using the MLN–EC axioms. Finally, in Section 3.4 and 3.5 the clause creation/evaluation and weight learning procedure are presented respectively.

3.1 Extracting Templates from Axioms

OSLα begins by partitioning the background knowledge into a set of axioms $\mathcal{A}$ and a set of domain-dependent definitions $\mathcal{B}$, that is, the rest of the formulas. Each axiom $\alpha \in \mathcal{A}$ should contain exactly one so-called *template predicate* as well as at least one *query predicate* representing the CEs we are trying to recognize. These are the desired properties required for OSLα in order to operate. Furthermore, axioms must not contain free variables, meaning variables only appearing in a single predicate.

In the case of the Event Calculus, $\mathcal{A}$ contains the four axioms of MLN–EC (2.3)–(2.6) that designate the properties of the formalism. Then, `HoldsAt` $\in \mathcal{Q}$ are the *query predicates* and `InitiatedAt`, `TerminatedAt` $\in \mathcal{P}$ are the *template predicates*. Those latter predicates specify the conditions under which a CE starts and stops being recognized respectively. They form the target CE patterns that we want to learn. Therefore, these four axioms of MLN–EC can be used to define a template over all possible structures and guide the search in selecting clauses following the desired properties of the Event Calculus. By exploiting the information of this template, the algorithm does not need to search over time sequences during relational pathfinding, but only needs to find explanations for the template predicates over the current time-point, in the form of definite clauses. Following the work of Skarlatidis et al. [2015], we perform circumscription by *predicate completion*, a syntactic transformation where formulas are translated into stronger ones. Predicate completion is applied to the `InitiatedAt` and `TerminatedAt` predicates. Finally, we also eliminate the `InitiatedAt` and `TerminatedAt` predicates by exploiting the equivalences resulting from predicate completion [Mueller, 2008]. Formally speaking, the algorithm only needs to search for definite clauses having the following form:

$$\texttt{InitiatedAt}(f, t) \Leftarrow \text{body}$$
$$\texttt{TerminatedAt}(f, t) \Leftarrow \text{body}$$

The body of these definitions is a conjunction of n literals $\ell_1 \wedge \cdots \wedge \ell_n$, which is very convenient, because it can be seen as a variabilized hypergraph path as we shall explain below. Given a set of axioms $\mathcal{A}$, we further partition it into templates. Each template $\mathcal{T}_i$ contains axioms having identical Cartesian product of domain types over their template predicate variables. For instance, MLN–EC axioms (2.3)–(2.6) should all belong to one template because `InitiatedAt` and `TerminatedAt` both have joint domain $\mathcal{F} \times \mathcal{T}$. Each of these resulting templates $\mathcal{T}_i$ is used during relational pathfinding in order to constrain the search space into specific bodies for the definite clauses.

3.2 Hypergraph and Relational Pathfinding

Following the procedure of OSL, at each step t the algorithm receives an example $\mathbf{x}_t$ representing the evidence, it produces the predicted label $\mathbf{y}_t^P = \text{argmax}_{\mathbf{y}\in\mathcal{Y}}\langle \mathbf{w}, \mathbf{n}(\mathbf{x}_t, \mathbf{y})\rangle$, and then receives the true label $\mathbf{y}_t$. Given both $\mathbf{y}_t$ and $\mathbf{y}_t^P$, in order to discover clauses that separate $\mathbf{y}_t$ from $\mathbf{y}_t^P$, it finds all ground atoms that are in $\mathbf{y}_t$ but not in $\mathbf{y}_t^P$ denoted as $\Delta y_t = \mathbf{y}_t \setminus \mathbf{y}_t^P$. That means Δy_t contains the false positives and false negatives resulted from inference. Then, it searches the ground-truth possible world ($\mathbf{x}_t$,$\mathbf{y}_t$), namely the training example of the current step t, for clauses specific to the axioms defined in the background knowledge using the constructed templates. In contrast to OSL, OSLα considers all misclassified (false positives/negatives) ground atoms instead of the true ones (false negatives), and searches the ground-truth possible world for clauses.

Simple Derived Events	**Supervision of Composite Events**
...	...
HappensAt(walking(ID_1), 99)	
HappensAt(walking(ID_2), 99)	
OrientationMove(ID_1, ID_2, 99)	HoldsAt(move(ID_1, ID_2), 100)
Close(ID_1, ID_2, 34, 99)	
Next(*99*, *100*)	
...	...
...	...
AUXwalking($WalkingID_1$, ID_1)	
AUXwalking($WalkingID_2$, ID_2)	
AUXmove($MoveID_1ID_2$, ID_1, ID_2)	
AUXmove($MoveID_2ID_1$, ID_2, ID_1)	
HappensAt($WalkingID_1$, 99)	
HappensAt($WalkingID_2$, 99)	
OrientationMove(ID_1, ID_2, 99)	HoldsAt($MoveID_1ID_2$, 100)
Close(ID_1, ID_2, 34, 99)	
Next(*99*, *100*)	
...	...

TABLE 3.1: Example training set for move CE. The first column is composed of a narrative of SDEs, while the second column contains the CE annotation in the form of ground HoldsAt predicates.

In order to discover useful clauses specific to a set of wrongly predicted atoms, we employ relational pathfinding, which considers a training example $\mathcal{D}$ as a hypergraph having constants as nodes and true ground atoms as hyperedges, connecting the nodes appearing as its arguments. Hyperedges are a generalization of edges connecting any number of nodes. It then searches the hypergraph for paths that connect the arguments of an input literal. Consider, for example, a training example $\mathcal{D}_t$ at step t of SDEs and CEs as shown

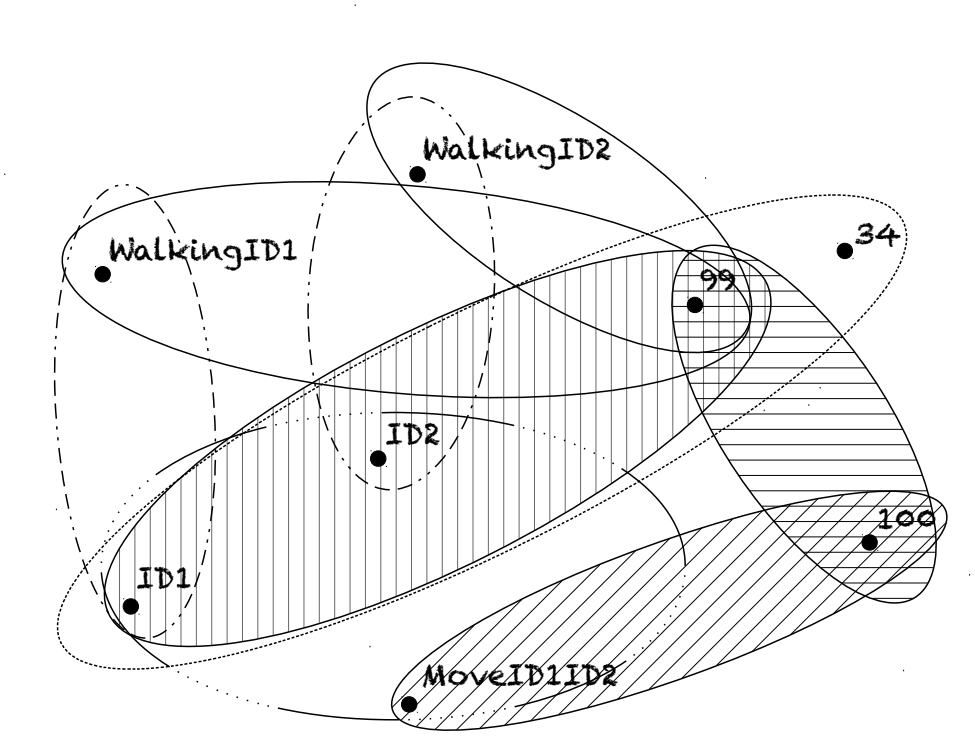

FIGURE 3.2: Hypergraph for the training set of Table 3.1.Continuous line ellipses represent the `HappensAt` predicates, dashed line `AUXwalking`, line followed by dots `AUXmove`, fine-dotted `Close`, horizontal line filled `Next`, vertical line filled `OrientationMove` and the diagonal line filled `HoldsAt`.

in Table 3.1. In the top part of the table the training set is presented using functions and in the bottom part using auxiliary predicates. Each auxiliary predicate models the behavior of a function. For example the predicate `AUXwalking` matches the return values of function `walking` with the domain of `Id`. Auxiliary predicates are required in order to indirectly include functions into the hypergraph.

The equivalent hypergraph representing the training example of Table 3.1 is presented in Figure 3.2, where each ellipse is a hyperedge and each dot is a constant node. Starting from each wrongly predicted ground atom in Δy_t, relational pathfinding searches for all paths (up to length l) connecting the arguments of the given atom. In our case, these ground atoms correspond to the wrongly predicted CEs. A path of hyperedges corresponds to a conjunction of true ground atoms connected by their arguments and can be generalized into conjunction of variabilized literals. For example consider the training set of Table 3.1. If the predicted label y_t^P says that $\texttt{HoldsAt}(MoveID_1\,ID_2,\,100)$ is false, then is considered a wrongly predicted atom and therefore the hypergraph should be searched for paths. Below, we present two paths found by searching the hypergraph of Figure 3.2 for paths up to length $l = 7$ for this misclassified CE. These paths are also presented in Figure 3.3 with highlighted hyperedges. Obviously there are many other possible paths that can be found.

$$\{\texttt{HoldsAt}(MoveID_1\,ID_2,\,100),\,\texttt{Next}(99,\,100),\,\texttt{HappensAt}(WalkingID_1,\,99),\,\texttt{HappensAt}(WalkingID_2,\,99),\,\texttt{AUXwalking}(WalkingID_1, ID_1),\,\texttt{AUXwalking}(WalkingID_2, ID_2),\,\texttt{AUXmove}(MoveID_1ID_2, ID_1, ID_2)\} \tag{3.1}$$

$$\{\texttt{HoldsAt}(MoveID_1\,ID_2,\,100),\,\texttt{Next}(99,\,100),\,\texttt{Close}(ID_1,\,ID_2,\,34,\,99),\,\texttt{AUXmove}(MoveID_1ID_2, ID_1, ID_2)\} \tag{3.2}$$

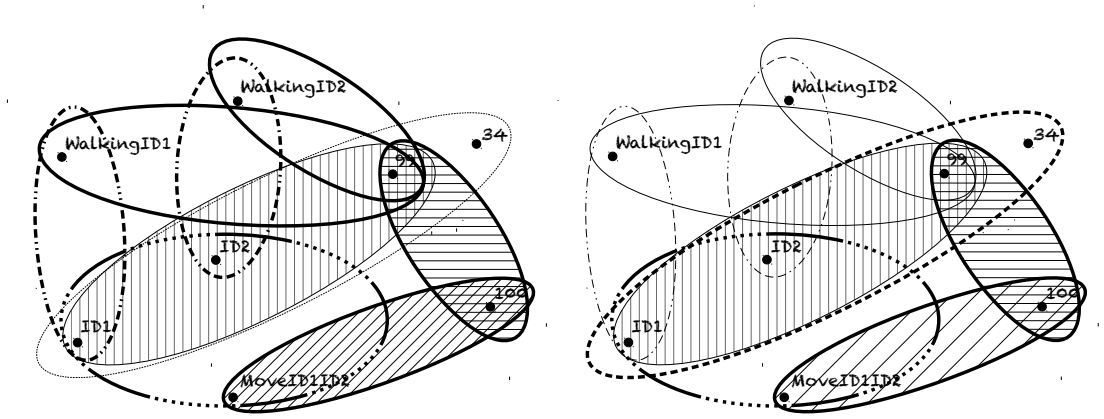

FIGURE 3.3: Hypergraph paths discovered by relational pathfinding

To speed up relational pathfinding, we employ mode declarations to constrain the search for paths, which represent the body of the definite clauses defined by the template predicates appearing in the axioms of the background knowledge. Mode declarations are a form of language bias that constrain the search for definite clauses. Since we want to constrain the space of paths, we use a variant of the path mode declarations introduced by Huynh and Mooney [2011b]. Formally speaking, declaration $\texttt{modep}(r, p)$ has two components: a recall number $r \in \mathbb{N}_0$, and an atom p whose arguments are place-markers optionally preceding the symbol $\#$. A place-marker is either $+$ (input), $-$ (output), or $.$ (ignore). The symbol $\#$ preceding each place-marker specifies that this particular predicate argument will remain constant after the generalization of the path. For many tasks it is critical to have clauses specific to particular constants. The recall number r limits the number of appearances of the predicate p in a path to r. These place-markers restrict the search of relational pathfinding. A ground atom is only added to a path, if one of its arguments has previously appeared as 'input' or 'output' arguments in the path and all of its 'input' arguments are 'output' arguments of previous atoms.

Furthermore, in order for our algorithm to be able to handle functions, we also introduce mode declarations for functions, defined as $\texttt{modef}(r, p)$ and having exactly the same functionality as described above. The only difference is that functions have a return type and the position for this type cannot be declared in the mode declaration; it is

always assumed as 'input'. The intuition behind this is that a function always returns a constant value belonging to some other function or predicate as an argument. Therefore, the return value must appear before in the path. Below we present a couple of mode declarations for both predicates and functions:

$$\texttt{modep}(1, \texttt{HappensAt}(-, +)) \quad \texttt{modef}(1, \texttt{move}(-, -))$$

The above mode declarations require that a legal path contains at most one ground atom of each of the predicates `HappensAt` and at most one ground function `move`. Moreover, the first argument of `HappensAt` and all arguments of `move` are 'output' arguments; the second argument of `HappensAt` is an 'input' argument. The 'input' mode constrains the position constant in `HappensAt` atoms to have appeared in previous atoms in a path.

Algorithm 1 HG($\mathcal{D}$, modes, $\mathcal{P}$)

Input: $\mathcal{D}$: training example, modes: mode declarations, $\mathcal{P}$: set of *template predicates*
Output: HG: hypergraph

1: **for all** constant $c \in \mathcal{D}$ **do**
2: $\quad HG[c] = \emptyset$
3: **for all** true ground atom $\mathrm{p}(c_1, \ldots, c_r) \in \mathcal{D}$ & $\mathrm{p} \notin \mathcal{P}$ **do**
4: $\quad$ **for all** constant $c_i \in \{c_1, \ldots, c_r\}$ **do**
5: $\quad\quad$ **if** $\mathrm{isInputOrOutputVar}(c_i, \mathrm{modes})$ **then**
6: $\quad\quad\quad HG[c_i] = HG[c_i] \cup \{\mathrm{p}(c_1, \ldots, c_r)\}$
return HG

Algorithm 1 presents the pseudocode for constructing a reduced hypergraph from a training example $\mathcal{D}$ based on mode declarations, by only allowing input and output nodes. There is no point in constructing the entire search space, because only the portion of it defined by the mode declarations will be eventually searched. Note that template predicates are not added in the hypergraph, because they are not allowed to appear in the body of the definite clause. Moreover, in order to search for functions, before the hypergraph is constructed, all functions in the domain are converted into auxiliary predicates (see Table 3.1 for example) and their corresponding mode declarations are converted into predicate modes. For example, the function `move` will be converted into the auxiliary predicate `AUXmove` having arity 3, because it also incorporates the return value of the function and the corresponding mode declaration $\texttt{modef}(1, \texttt{move}(-, -))$ will be converted into $\texttt{modep}(1, \texttt{AUXmove}(+, -, -))$. By performing this conversion, the functions can be included in the hypergraph as auxiliary predicates and added to paths during the search procedure.

3.3 Template Guided Search

Starting from each wrongly predicted ground atom $q(c_1, \ldots, c_n) \in \Delta y_t$, we use the templates $\mathcal{T}_i$ constructed at the initial steps of the algorithm in order to find the corresponding ground template predicates for which the axioms belonging in the template $\mathcal{T}_i$ are satisfied by the current training example. The algorithm considers each axiom $\alpha \in \mathcal{A}$ in turn and checks whether the desired properties, presented in Section 3.1, hold. Assume, for example, that one of these axioms is 2.3:

$$\texttt{HoldsAt}(f,\, t{+}1) \Leftarrow \\ \texttt{Next}(t,\, t{+}1) \wedge \texttt{InitiatedAt}(f,\, t)$$

and we have given the wrongly predicted ground atom $\texttt{HoldsAt}(CE,\, T_4)$ (false negative). We would substitute the constants of the given atom q into the axiom. The result of the substitution on the above rule will be the following partially grounded axiom:

$$\texttt{HoldsAt}(CE,\, T_4) \Leftarrow \\ \texttt{Next}(t,\, T_4) \wedge \texttt{InitiatedAt}(CE,\, t)$$

If after the substitution there are no variables left in the template predicate of the axiom, it adds the grounded template predicate into the initiation set $\mathcal{I}$ and moves to the next axiom. Otherwise, it searches for all literals in the axiom sharing variables with the template predicate. For these literals, it searches the training data for all jointly ground instantiations among those satisfying the axiom. Then, for each of them, it substitutes the remaining variables of the template predicate and adds them into the initiation set. In this example, `InitiatedAt` has one remaining variable t and the only literal sharing this variable is `Next`. Thus, we search for all ground instantiations of `Next` in the training data $\mathcal{D}$ that satisfy the axiom and substitute their constants into the partially grounded axiom. Because t represents time-points and the predicate `Next` describes sequence of time-points, there will be only one true ground atom in the training data $\mathcal{D}$ having the constant T_3. The same applies for axioms 2.5 and 2.6 determining the termination conditions and false positives. Algorithm 2 presents the pseudocode for extracting these ground template predicates.

For each grounded template predicate returned by Algorithm 2, the mode-guided relational pathfinding, presented in Algorithm 3, is used to search the constructed hypergraph for an appropriate body. It recursively adds to the path ground atoms or hyperedges

Algorithm 2 InitialSet(q, $\mathcal{D}$, $\mathcal{T}$)

Input: q: misclassified ground atom, $\mathcal{D}$: training example, $\mathcal{T}$: path template
Output: $\mathcal{I}$: a set of grounded template predicates

$\mathcal{I} = \emptyset$ ▷ The set of ground template predicates used to initiate the search
for all axiom $\alpha \in \mathcal{T}$ **do**
  if $\exists$ literal $\ell \in \alpha$: $\text{signature}(\ell) = \text{signature}(q)$ & $\text{isPositive}(\ell) == \text{isTrue}(g, \mathcal{D})$ **then**
    θ-substitute $\beta = \alpha\theta$ where $\theta = \{\text{variables}(\ell) \rightarrow \text{constants}(q)\}$
    $\tau = \text{templateAtom}(\beta)$
    if $\exists$ variable $v \in \tau$ **then**
      $\mathcal{L} = \emptyset$
      if $\exists$ variable $v \in \tau \wedge \exists$ literal $\ell \in \beta : v \in \ell$ **then**
        $\mathcal{L} = \mathcal{L} \cup \ell$
      for all literal $\ell \in \mathcal{L}$ **do**
        $\mathbf{T}_l = \{c_1, \ldots, c_n\} \in \mathcal{D} : \ell(c_1, \ldots, c_n) \Rightarrow \top$
      for all row $r = \{c_1, \ldots, c_n\} \in \bowtie_{\ell=1}^{|\mathcal{L}|} \mathbf{T}_l$ **do**
        θ-substitute $\gamma = \beta\theta$ where $\theta = \{variables(\beta) \Rightarrow r\}$
        $\mathcal{I} = \mathcal{I} \cup \text{templateAtom}(\gamma)$
    else
      $\mathcal{I} = \mathcal{I} \cup \tau$
return $\mathcal{I}$

that satisfy the mode declarations. The search terminates when the path reaches a specified maximum length or when no new hyperedges can be added. The algorithm stores all paths encountered during the search.

Algorithm 3 ModeGuidedSearch(curPath, $\mathcal{C}$, HG, mode, maxLength, paths)

Output: paths: a set of paths

if |curPath| $<$ maxLength **then**
  for all constant $c \in \mathcal{C}$ **do**
    for all $\text{p}(c_1, \ldots, c_r) \in HG[c]$ **do**
      if canAdd(p, curPath, mode) **then**
        if curPath $\notin$ paths **then**
          curPath = curPath $\cup\{\text{p}(c_1, \ldots, c_r)\}$
          paths = paths $\cup\{p(c_1, \ldots, c_r)\}$
          $\mathcal{C}' = \emptyset$
          for all $c_i \in \{c_1, \ldots, c_r\}$ **do**
            if $c_i \notin \mathcal{C}$ & isInputOrOutputVar(c_i, mode) **then**
              $\mathcal{C} = \mathcal{C} \cup \{c_i\}$
              $\mathcal{C}' = \mathcal{C}' \cup \{c_i\}$
          ModeGuidedFindPath(curPath,$\mathcal{C}$,HG,mode,maxLength,paths)
          curPath = curPath $\setminus$p
          $\mathcal{C} = \mathcal{C} \setminus \mathcal{C}'$

By employing this procedure the hypergraph is essentially pruned in order to contain only ground atoms explaining the template predicates. Consider the hypergraph presented in Figure 3.2. By exploiting the Event Calculus axioms the hypergraph is pruned and only

need to contain predicates explaining the `InitiatedAt` and `TerminatedAt` as presented in Figure 3.4. Therefore the paths (3.1) and (3.2) are pruned by removing `Next` and `HoldsAt` predicates, resulting into the paths (3.3) and (3.4). The pruning resulting from the template guided search is essential to learn Event Calculus, because the search space becomes independent of time.

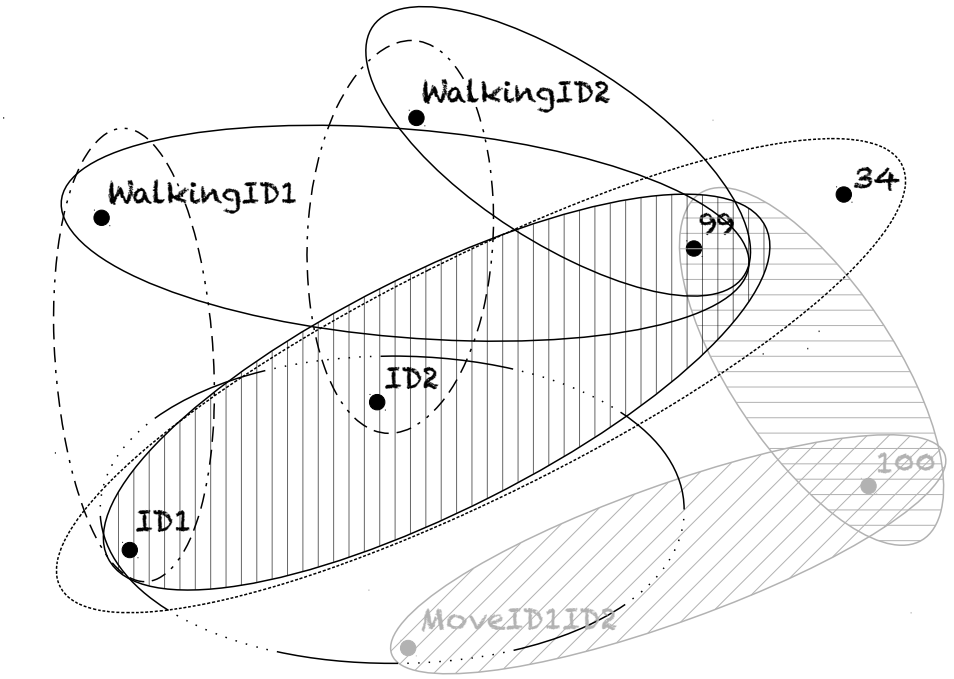

FIGURE 3.4: Pruned hypergraph

$$\big\{\texttt{InitiatedAt}(MoveID_1\,ID_2,\,99),\,\texttt{HappensAt}(\,WalkingID_1,\,99),\\ \texttt{HappensAt}(\,WalkingID_2,\,99),\,\texttt{AUXwalking}(WalkingID_1, ID_1),\\ \texttt{AUXwalking}(WalkingID_2, ID_2),\,\texttt{AUXmove}(MoveID_1ID_2, ID_1, ID_2)\big\} \tag{3.3}$$

$$\big\{\texttt{InitiatedAt}(MoveID_1\,ID_2,\,99),\,\texttt{Close}(ID_1,\,ID_2,\,34,\,99),\\ \texttt{AUXmove}(MoveID_1ID_2, ID_1, ID_2)\big\} \tag{3.4}$$

3.4 Clause Creation and Evaluation

In order to generalize paths into first-order clauses, we follow three steps. First, we replace each constant c_i in a conjunction with a variable v_i, except for those declared constant in the mode declarations. Then, these conjunctions are used as a body to form definite clauses using the template predicate present in each path as head. The auxiliary

predicates are converted back into functions. Therefore, from the paths presented above, the following definite clauses will be created:

$$\begin{aligned}&\texttt{InitiatedAt}(\texttt{move}(id_1, id_2),\, t) \Leftarrow \\ &\qquad \texttt{HappensAt}(\texttt{walking}(id_1),\, t) \wedge \\ &\qquad \texttt{HappensAt}(\texttt{walking}(id_2),\, t)\end{aligned} \tag{3.5}$$

$$\begin{aligned}&\texttt{InitiatedAt}(\texttt{move}(id_1, id_2),\, t) \Leftarrow \\ &\qquad \texttt{Close}(id_1,\, id_2,\, 34,\, t)\end{aligned} \tag{3.6}$$

The first definition says that the CE `move` is initiated for a person id_1 and a person id_2 at t when the event `walking` happens for both of them. Similarly the second definition says that `move` is initiated when id_1 and id_2 are close enough, which does not make sense. These definite clauses can be used together with the axioms defined in the background knowledge in order to eliminate all the template predicates by exploiting the equivalences resulting from predicate completion. After the elimination process all resulting formulas are converted into clausal normal form (CNF), since that is the form used by the inference algorithms. Therefore, the resulting set of clauses is independent of the template predicates. Evaluation must take place for each clause c individually. The difference in the number of true groundings of c in the ground-truth world $(\mathbf{x}_t, \mathbf{y}_t)$ and the predicted world $(\mathbf{x}_t, \mathbf{y}_t^P)$ is computed. Then, the only clauses whose difference in the number of groundings is greater than or equal to a predefined threshold μ will be added to the existing MLN:

$$\Delta n_c = n_c(\mathbf{x}_t, \mathbf{y}_t) - n_c(\mathbf{x}_t, \mathbf{y}_t^P)$$

The intuition behind this measure is to keep clauses whose coverage of the ground-truth world is different from the one induced by the clauses already learnt. Subsequently, it may be necessary to perform again predicate completion and template predicate elimination, because the resulting set of formulas returned by the circumscription changes entirely if any one definite clause is removed during evaluation. To illustrate these changes in the resulting hypothesis, consider the domain-dependent definitions of `move` – i.e., rules (3.5)–(3.6). After predicate completion, these rules will be replaced by the following formula:

$$\begin{aligned}&\texttt{InitiatedAt}(\texttt{move}(id_1, id_2),\, t) \Leftrightarrow \\ &\qquad \big(\texttt{HappensAt}(\texttt{walking}(id_1),\, t) \wedge \\ &\qquad \texttt{HappensAt}(\texttt{walking}(id_2),\, t)\big) \bigvee \\ &\qquad \texttt{Close}(id_1,\, id_2,\, 34,\, t)\end{aligned} \tag{3.7}$$

The resulting rule (3.7) defines all conditions under which the `move` CE is initiated. Based on the equivalence in formula (3.7), the domain-independent axiom (2.3) of MLN–EC automatically produces the following template predicates free (i.e, `InitiatedAt`, `TerminatedAt`) rules:

$$\begin{aligned}\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t{+}1) \Leftarrow\\ \texttt{HappensAt}(\texttt{walking}(id_1),\, t) \wedge\\ \texttt{HappensAt}(\texttt{walking}(id_2),\, t)\end{aligned} \tag{3.8}$$

$$\begin{aligned}\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t{+}1) \Leftarrow\\ \texttt{Close}(id_1,\, id_2,\, 34,\, t)\end{aligned} \tag{3.9}$$

Similarly, the inertia axiom (2.6) produces:

$$\begin{aligned}\neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t{+}1) \Leftarrow\\ \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t) \wedge\\ \neg\Big(\big(\texttt{HappensAt}(\texttt{walking}(id_1),\, t) \wedge\\ \texttt{HappensAt}(\texttt{walking}(id_2),\, t)\big) \bigvee\\ \texttt{Close}(id_1,\, id_2,\, 34,\, t)\Big)\end{aligned} \tag{3.10}$$

Consider now, that during the evaluation process the definite clause (3.9) yields a score less than μ and therefore it is discarded. Then, the resulting hypothesis is reduced to the following rules:

$$\begin{aligned}\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t{+}1) \Leftarrow\\ \texttt{HappensAt}(\texttt{walking}(id_1),\, t) \wedge\\ \texttt{HappensAt}(\texttt{walking}(id_2),\, t)\end{aligned} \tag{3.11}$$

$$\begin{aligned}\neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t{+}1) \Leftarrow\\ \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t) \wedge\\ \neg\big(\texttt{HappensAt}(\texttt{walking}(id_1),\, t) \wedge\\ \texttt{HappensAt}(\texttt{walking}(id_2),\, t)\big)\end{aligned} \tag{3.12}$$

3.5 Weight Learning

The weights of all retained clauses are learnt or updated using the AdaGrad online learner described in Section 2.2.4. As stated in the previous section the circumscription of predicate completion depends upon the set of definite clauses, in the sense that if

that set changes then the resulting clauses may be different. At each step t of OSLα the definite clauses are updated by adding new clauses found during the hypergraph search procedure and therefore the resulting set of clauses $\mathcal{C}_t$ is different from the set $\mathcal{C}_{t-1}$. In order for AdaGrad to be able to apply weight updates in a constantly changing theory, OSLα searches for clauses in the current theory $\mathcal{C}_t$ that are θ-subsumed [Raedt, 2008] by a clause in the previous theory and inherit its weight. The intuition is that if a clause is θ-subsumed by another clause then is at least as general as the latter one. All others are considered new and their weights are set to an initial value close to zero. In this way the already learned weight values are transfered to the next step of the procedure. To illustrate the procedure consider a set of definite clauses learned in step t:

$$D_t = \begin{cases} \texttt{InitiatedAt}(\texttt{move}(id_1, id_2),\, t) \Leftarrow \\ \qquad\qquad \texttt{HappensAt}(\texttt{walking}(id_1),\, t) \wedge \\ \qquad\qquad \texttt{HappensAt}(\texttt{walking}(id_2),\, t) \\ \texttt{TerminatedAt}(\texttt{move}(id_1, id_2),\, t) \Leftarrow \\ \qquad\qquad \texttt{HappensAt}(\texttt{inactive}(id_1),\, t) \wedge \\ \qquad\qquad \texttt{HappensAt}(\texttt{active}(id_2),\, t) \end{cases}$$

By performing predicate completion upon the set D_t and using the domain-independent axioms of MLN–EC to flatten the produced rules, thus eliminating the template predicates, the following hypothesis arises:

$$\Sigma_t = \begin{cases} \texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t{+}1) \Leftarrow & (3.13) \\ \quad \texttt{HappensAt}(\texttt{walking}(id_1),\, t) \wedge \texttt{HappensAt}(\texttt{walking}(id_2),\, t) & \\ \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t{+}1) \Leftarrow & (3.14) \\ \quad \texttt{HappensAt}(\texttt{inactive}(id_1),\, t) \wedge \texttt{HappensAt}(\texttt{active}(id_2),\, t) & \end{cases}$$

$$\Sigma'_t = \begin{cases} \texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t{+}1) \Leftarrow & \\ \quad \texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t) \wedge & (3.15) \\ \quad \neg\big(\texttt{HappensAt}(\texttt{inactive}(id_1),\, t) \wedge \texttt{HappensAt}(\texttt{active}(id_2),\, t)\big) & \\ \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t{+}1) \Leftarrow & \\ \quad \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\, t) \wedge & (3.16) \\ \quad \neg\big(\texttt{HappensAt}(\texttt{walking}(id_1),\, t) \wedge \texttt{HappensAt}(\texttt{walking}(id_2),\, t)\big) & \end{cases}$$

These rules are further transformed into CNF, in order to be able to carry out inference, but for the purposes of demonstration we keep them in their initial form. The rules in (3.13) - (3.16) are separated into two subsets. The former set Σ_t contains specialized definitions of axioms (2.3) and (2.5), specifying that a fluent holds (or does not hold) when its initiation (or termination) conditions are met. The latter set Σ'_t contains specialized definitions of the inertia axioms (2.4) and (2.6), specifying whether a specific fluent continues to hold or not at any instance of time. Both sets are passed onto the

weight learning module in order for their weights to be estimated. Let's suppose that weight learning maps these rules onto the following weight values:

$$W_t = \begin{cases} \mathbf{0.56}\ \texttt{HoldsAt}(\texttt{move}(id_1, id_2),\ t{+}1) \Leftarrow & (3.17) \\ \quad \texttt{HappensAt}(\texttt{walking}(id_1),\ t) \wedge \texttt{HappensAt}(\texttt{walking}(id_2),\ t) & \\ \mathbf{0.71}\ \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\ t{+}1) \Leftarrow & (3.18) \\ \quad \texttt{HappensAt}(\texttt{inactive}(id_1),\ t) \wedge \texttt{HappensAt}(\texttt{active}(id_2),\ t) & \\ \mathbf{0.47}\ \texttt{HoldsAt}(\texttt{move}(id_1, id_2),\ t{+}1) \Leftarrow & (3.19) \\ \quad \texttt{HoldsAt}(\texttt{move}(id_1, id_2),\ t) \wedge & \\ \quad \neg\big(\texttt{HappensAt}(\texttt{inactive}(id_1),\ t) \wedge \texttt{HappensAt}(\texttt{active}(id_2),\ t)\big) & \\ \mathbf{0.24}\ \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\ t{+}1) \Leftarrow & (3.20) \\ \quad \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\ t) \wedge & \\ \quad \neg\big(\texttt{HappensAt}(\texttt{walking}(id_1),\ t) \wedge \texttt{HappensAt}(\texttt{walking}(id_2),\ t)\big) & \end{cases}$$

Then consider that in the next learning step t' of OSLα another definite clause is learned and the set of definite clauses is expanded into following hypothesis:

$$D_{t'} = \begin{cases} \texttt{InitiatedAt}(\texttt{move}(id_1, id_2),\ t) \Leftarrow \\ \qquad \texttt{HappensAt}(\texttt{walking}(id_1),\ t) \wedge \\ \qquad \texttt{HappensAt}(\texttt{walking}(id_2),\ t) \\ \texttt{TerminatedAt}(\texttt{move}(id_1, id_2),\ t) \Leftarrow \\ \qquad \texttt{HappensAt}(\texttt{inactive}(id_1),\ t) \wedge \\ \qquad \texttt{HappensAt}(\texttt{active}(id_2),\ t) \\ \texttt{TerminatedAt}(\texttt{move}(id_1, id_2),\ t) \Leftarrow \\ \qquad \texttt{HappensAt}(\texttt{exit}(id_1),\ t) \end{cases}$$

Similarly to D_t, by applying predicate completion to $D_{t'}$ and flattening the rules using the domain-independent axioms of MLN–EC a different hypothesis arises:

$$\Sigma_{t'} = \begin{cases} \texttt{HoldsAt}(\texttt{move}(id_1, id_2),\ t{+}1) \Leftarrow & (3.21) \\ \quad \texttt{HappensAt}(\texttt{walking}(id_1),\ t) \wedge \texttt{HappensAt}(\texttt{walking}(id_2),\ t) & \\ \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\ t{+}1) \Leftarrow & (3.22) \\ \quad \texttt{HappensAt}(\texttt{inactive}(id_1),\ t) \wedge \texttt{HappensAt}(\texttt{active}(id_2),\ t) & \\ \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2),\ t{+}1) \Leftarrow & (3.23) \\ \quad \texttt{HappensAt}(\texttt{exit}(id_1),\ t) & \end{cases}$$

$$\Sigma'_{t'} = \begin{cases} \texttt{HoldsAt}(\texttt{move}(id_1, id_2), t{+}1) \Leftarrow \\ \quad \texttt{HoldsAt}(\texttt{move}(id_1, id_2), t) \wedge \\ \quad \neg\big(\texttt{HappensAt}(\texttt{inactive}(id_1), t) \wedge \texttt{HappensAt}(\texttt{active}(id_2), t)\big) & (3.24) \\ \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2), t{+}1) \Leftarrow \\ \quad \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2), t) \wedge \\ \quad \neg\big(\,(\texttt{HappensAt}(\texttt{walking}(id_1), t) \wedge \texttt{HappensAt}(\texttt{walking}(id_2), t)) \vee \\ \quad \texttt{HappensAt}(\texttt{exit}(id_1), t)\big) & (3.25) \end{cases}$$

Note that in the set $\Sigma_{t'}$ another rule (3.23) has appeared and in the set $\Sigma'_{t'}$ the rule (3.25) changed by incorporating another literal. Therefore, in order to refine the weights of the current theory at step t' a mapping of the previous learned weights onto the current theory is required, so that the already learned values are retained. To achieve that, OSLα searches for clauses in the current theory that are θ-subsumed by a clause in the previous theory and inherit its weight. In our example, rules (3.21), (3.22), and (3.24) are identical to (3.13), (3.14), (3.15) respectively and consequently they inherit their weights. Rule (3.23) is completly new and its weight is set to an initial value. Finally, rule (3.25) is θ-subsumed by (3.16) and therefore should inherit its weight. Thereafter, weight learning takes place in order to refine the inherited weights according to the current training data. The intuition behind this is that clauses changing their structure by introducing new literals are preferred to have an initial weight value close to a similar structured clause.

Moreover, we can treat the rules in sets Σ and Σ' as either hard or soft constraints and modify the behavior of MLN–EC as stated by Skarlatidis [2014]; Skarlatidis et al. [2015]. In order to do this, we only need to define the corresponding axioms of the MLN–EC as either soft or hard constraints in the background knowledge. By doing so, the corresponding sets of Σ and Σ' will produce either soft or hard constraint clauses, during predicate completion and elimination, depending on the axiom used to produce each clause during predicate completion. If the axiom has an infinite weight, then the corresponding clauses created by it will also have an infinite weight and therefore it cannot be changed during weight learning.

Finally, at the end of the OSLα learning we can choose to remove those clauses whose weights are less than a predefined threshold ξ in order to speed-up inference. By doing so and finding a good ξ, the resulting hypothesis can be pruned significantly and the accuracy, although reduced, remains in acceptable levels. We present below the complete procedure of OSLα.

Algorithm 4 OSLa($\mathcal{C}$,$\mathcal{P}$,modes,maxLength,μ,λ,η,δ)

Input: $\mathcal{KB}$: initial knowledge base
1: $\mathcal{P}$: template predicates
2: modes: mode declarations
3: maxLength: maximum number of hyperedges in a path
4: μ: evaluation threshold
5: λ, η, δ: AdaGrad parameters
6: Partition $\mathcal{KB}$ into $\mathcal{A}$ and $\mathcal{B}$
7: Create templates $\boldsymbol{\mathcal{T}}$ from $\mathcal{A}$
8: Initialize resulting theory $\mathcal{R}_0 = \mathcal{B}$
9: Initialize weight vector $\mathbf{w}_0 = \text{initialValue}$
10: **for** $t = 1$ to T **do**
11: Receive an instance $\mathbf{x}_t$
12: Predict $\mathbf{y}_t^P = \text{argmax}_{\mathbf{y}\in\boldsymbol{\mathcal{Y}}}\langle \mathbf{w}, \mathbf{n}(\mathbf{x}_t, \mathbf{y})\rangle$
13: Receive the correct target $\mathbf{y}_t$
14: Compute $\Delta y_t = \mathbf{y}_t \setminus \mathbf{y}_t^P$
15: **if** $\Delta y_t \neq \emptyset$ **then**
16: $HG = \text{HG}\big((\mathbf{x}_t, \mathbf{y}_t), \text{modes}, \mathcal{P}\big)$
17: paths= $\emptyset$
18: **for all** wrogly predicted query atom $\text{q} \in \Delta y_t$ **do**
19: **for all** template $\mathcal{T}_i \in \boldsymbol{\mathcal{T}}$ **do**
20: $\mathcal{I} = \text{InitialSet}\big(\text{q}, (\mathbf{x}_t, \mathbf{y}_t), \mathcal{T}_i\big)$
21: $\mathcal{V} = \emptyset$
22: **for all** $\tau(c_1, \ldots, c_n) \in \mathcal{I}$ **do**
23: **for all** $c_i \in \{c_1, \ldots, c_n\}$ **do**
24: **if** $\text{isInputOrOutputVar}(c_i, \text{modes})$ **then**
25: $\mathcal{V} = \mathcal{V} \cup c_i$
26: $\text{ModeGuidedSearch}\big(\text{q}, \mathcal{V}, HG, \text{modes}, \text{maxLength}, \text{paths}\big)$
27: $\mathcal{D}_t = \text{CreateDefinitions}(\mathcal{D}_{t-1}, \text{paths}, \text{modes})$
28: $\mathcal{C}_t = \text{CreateClauses}(\mathcal{D}_t, \text{modes})$
29: Compute $\Delta\mathbf{n}_{\mathcal{C}_t}$
30: **for** $i = 1$ to $|\mathcal{C}_t|$ **do**
31: **if** $\Delta\mathbf{n}_{\mathcal{C}_{t,i}} \leq \mu$ **then**
32: Remove definite clause d_i corresponding to $\mathcal{C}_{t,i}$ from D_t
33: $\mathcal{R}_t = \mathcal{B} \cup \text{CreateClauses}(\mathcal{D}_t, \text{modes})$
34: Compute $\Delta\mathbf{n}_{\mathcal{R}_t}$
35: **for** $i = 1$ to $|\mathcal{R}_t|$ **do**
36: **if** $\exists\, c_{i,t-1} \in \mathcal{R}_{t-1},\ c_{i,t} \in \mathcal{R}_t : c_{i,t-1}\ \theta$-subsumes $c_{i,t}$ **then**
37: $w_{i,t}$=$w_{i,t-1}$
38: **else**
39: $w_{i,t}$=initialValue
40: $\text{AdaGrad}(\mathbf{w}_t, \Delta\mathbf{n}_{\mathcal{R}_t}, \lambda, \eta, \delta)$

3.6 Summary

We addressed the issue of exploiting the background knowledge axiomatization in order to effectively learn definitions inheriting the properties of MLN–EC. Specifically, the MLN–EC axioms are used for constructing templates constraining the space of possible structures defined by the training data. OSLα considers both types of wrongly predicted CEs (false positives and false negatives) and searches for structures explaining their occurrences. Furthermore, it uses mode declarations for both predicates and functions in the hypergraph to further guide the search procedure in finding useful features (i.e., clauses). It employs an online strategy in order to handle large training sets by iteratively refining the already learnt hypothesis in terms of both structure and weights. Due to predicate completion, a procedure of weight inheritance was introduced in order to retain the already learnt weight values between steps of the online procedure. We contributed the implementations of the aforementioned algorithms along with the implementations of ILP inference, max-margin weight learning as well as CDA and AdaGrad online learners to the LoMRF open-source project (see Section 2.2.5), resulting to a state-of-the-art MLN framework.

4 Experimental Evaluation

"It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong."
— Richard P. Feynman

In this chapter we evaluate our OSLα learning method in the domain of video activity recognition. We use the publicly available benchmark dataset of the CAVIAR project[1]. The aim of the experiments is to assess the effectiveness of OSLα in learning weighted relational structures used for recognizing complex activities that take place between multiple persons, by exploiting information about simple observed individual activities. This dataset comprises 28 surveillance videos, where each frame is annotated by human experts from the CAVIAR team on two levels. The first level contains *simple derived events* (SDEs) that concern activities of individual persons or the state of objects. The second level contains *composite event* (CE) annotations, describing the activities between multiple persons and/or objects, i.e., people meeting and moving together, leaving an object and fighting.

In Section 4.1 we briefly describe the dataset characteristics and the experimental setup for all of our experiments. Then in Section 4.2 we give an overview of the compared methods. In Sections 4.3 and 4.4 we present the results of weight learning and OSLα learning performance respectively, and finally in Section 4.5 we present conclusions about the experimental evaluation.

4.1 Experimental Setup

The input to the learning methods is a stream of SDEs along with the CE annotations. The SDEs are representing people walking, running, staying active, or inactive. The first and the last time that a person or an object is tracked are represented by the SDEs enter and exit. Additionally, the coordinates of tracked persons or objects are also taken

[1] http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1

into consideration in order to express qualitative spatial relations, e.g. two persons being relatively close to each other. The CE supervision indicates when each of the CE holds. Table 4.1 presents the structure of the training sequences. Each sequence is composed of input SDEs (ground `HappensAt`), precomputed spatial constraints between pairs of people (ground `Close`), as well as the corresponding CE annotations (ground `HoldsAt`). Negated predicates in the training sequence state that the truth value of the corresponding predicate is False.

Simple Derived Events	**Supervision of Composite Events**
...	...
`HappensAt`(`walking`(ID_1), 100)	
`HappensAt`(`walking`(ID_2), 100)	
`OrientationMove`(ID_1, ID_2, 100)	
`Close`(ID_1, ID_2, 24, 100)	
`Next`(*100*, *101*)	`HoldsAt`(`move`(ID_1, ID_2), 101)
...	...
`HappensAt`(`exit`(ID_1), 200)	
`HappensAt`(`walking`(ID_2), 200)	
¬`OrientationMove`(ID_1, ID_2, 200)	
¬`Close`(ID_1, ID_2, 24, 200)	
`Next`(*200*, *201*)	¬`HoldsAt`(`move`(ID_1, ID_2), 201)

TABLE 4.1: Training example for `move` CE. The first column is composed of a narrative of SDEs, while the second column contains the CE annotation in the form of ground `HoldsAt` predicates.

The definitions of the CEs `meet` and `move` we are using were developed in Artikis et al. [2010][2]. The former CE (`meet`) indicates that two persons are meeting and the latter (`move`) indicates they are moving together. These definitions take the form of common sense rules and describe the conditions under which a CE starts or ends (`InitiatedAt`, `TerminatedAt`). For example, when two persons are walking together with the same orientation, then moving together starts being recognized. Similarly, when the same persons walk away from each other, then moving together stops being recognized.

From the 28 videos of the CAVIAR dataset, we have extracted 19 sequences that are annotated with the `meet` and/or `move` CEs. The rest of the sequences in the dataset are ignored, as they do not contain positive examples of the target CE. Out of the 19 sequences, 8 are annotated with both `meet` and `move` activities, 9 are annotated only with `move` and 2 only with `meet`. The total length of the extracted sequences is 12869 frames. Each frame is annotated with the (non-)occurrence of a CE and is considered an example instance. The whole dataset contains a total of 25738 annotated example instances. There are 6272 example instances in which `move` occurs and 3722 in which `meet` occurs.

[2]The MLN–EC definitions and CAVIAR dataset can be found at www.iit.demokritos.gr/~anskarl/pub/mlnec/MLN-EC_CAVIAR-20130319-00_07_20.tar.bz2

Consequently, for both CEs the number of negative examples is significantly larger than the number of positive examples, specifically 19466 for `move` and 22016 for `meet`. The input consists of a sequence of SDEs, i.e., `active`, `inactive`, `walking`, `running`, `enter` and `exit`, the spatial constraints `Close` and `OrientationMove`, which was precomputed, and its truth value was provided as input, as well as the supervision for `meet` and `move`.

Following the work of Skarlatidis [2014]; Skarlatidis et al. [2015] we used three different inertia settings (`HI`, $\texttt{SI}^{\texttt{h}}$ and `SI`, see Table 4.2 for a description). In all three variants, all rules are always soft-constrained, except the inertia rules that, depending on the setting, may be either soft-or hard-constrained.

Settings	Description
`HI`	All inertia rules are hard-constrained.
$\texttt{SI}^{\texttt{h}}$	The inertia rules of `HoldsAt` are soft-constrained, while the rest remains hard-constrained.
`SI`	All inertia rules are soft-constrained.

TABLE 4.2: Variants of CAVIAR, using hard and soft inertia rules.

Throughout the experimental analysis, the evaluation results was obtained using the MAP inference of Huynh and Mooney [2009] and are presented in terms of True Positives (TP), False Positives (FP), False Negatives (FN), Precision, Recall and F_1 score. All reported experiment statistics are micro-averaged over the instances of recognized CEs using 10-fold cross validation over the 19 sequences. Details about the dataset are presented in Table 4.3. The experiments were performed on a computer having an Intel i7 4790@3.6GHz processor (4 cores and 8 threads) and 16GiB of RAM, running Apple OSX version 10.11.

total SDEs	63147
average SDEs per fold	56832
total `meet` positive CEs	3722
total `move` positive CEs	6272
average `meet` positive CEs	3350
average `move` positive CEs	5600

TABLE 4.3: CAVIAR statistics

4.2 The Methods Being Compared

We begin by evaluating the online weight learning methods using manual definitions. We compare their results against the batch max-margin learning method. Moreover, for comparison purposes, we also include in the experiments the results of the logic-based activity recognition method of Artikis et al. [2010], which we call here $\texttt{EC}_{\texttt{crisp}}$. The

latter employs a different variant of the Event Calculus, uses the same manual definitions of CEs and cannot perform probabilistic reasoning. Then, we present the experimental results of our OSLα learning and compare them against the results obtained by the manual definitions trained by the online weight learning and the pure logic-based method $\texttt{EC}_{\texttt{crisp}}$. Lastly, we also present the running times of OSL for the `meet` CE.

4.3 Weight Learning Performance

In this section both online max-margin (CDA) and AdaGrad algorithms presented in Section 2.2.4 are compared for all three inertia configurations of Table 4.2 against the batch max-margin learning and $\texttt{EC}_{\texttt{crisp}}$. The CE definitions of `meet` and `move` CEs are transformed by using the domain-independent axioms of MLN–EC and exploiting the equivalences resulting from predicate completion to eliminate `InitiatedAt`, `TerminatedAt` predicates. Then by applying CNF transformation each theory results into 23 and 44 clauses for `meet` and `move` respectively. Results of weight learning for both CEs are presented in Tables 4.4 and 4.5. As expected the batch max-margin weight learning yields the best overall accuracy due the fact that it uses all the data at once in order to estimate the optimal weights. AdaGrad is the second best choice as it yields more accurate results as opposed to CDA and $\texttt{EC}_{\texttt{crisp}}$ in the majority of the cases.

Method	TP	FP	FN	Precision	Recall	F_1 score
$\texttt{EC}_{\texttt{crisp}}$	3099	1413	523	0.6868	**0.8556**	0.7620
$\texttt{CDA}_{\texttt{HI}}$	1868	106	1754	**0.9463**	0.5157	0.6676
$\texttt{CDA}_{\texttt{SI}^\texttt{h}}$	1767	183	1855	0.9061	0.4878	0.6342
$\texttt{CDA}_{\texttt{SI}}$	1552	355	2070	0.8138	0.4284	0.5614
$\texttt{AdaGrad}_{\texttt{HI}}$	3099	1397	523	0.6892	**0.8556**	0.7634
$\texttt{AdaGrad}_{\texttt{SI}^\texttt{h}}$	3096	1187	526	0.7228	0.8547	0.7833
$\texttt{AdaGrad}_{\texttt{SI}}$	3099	1397	523	0.6892	**0.8556**	0.7634
$\texttt{MaxMargin}_{\texttt{HI}}$	3099	1397	523	0.6892	**0.8556**	0.7634
$\texttt{MaxMargin}_{\texttt{SI}^\texttt{h}}$	2946	260	676	0.9189	0.8133	**0.8629**
$\texttt{MaxMargin}_{\texttt{SI}}$	3009	809	613	0.7673	0.4770	0.5883

TABLE 4.4: Weight learning accuracy of the `meet` CE

Note that the $\texttt{SI}^\texttt{h}$ inertia setting yields the best results for each individual method. In this setting, the inertia rule of `HoldsAt` remains soft-constrained. As a result, the probability of a CE tends to decrease, even when the required termination conditions are not met. Therefore, if the probability cannot be reinforced by a re-initiation then $\texttt{SI}^\texttt{h}$ setting is very useful [Skarlatidis, 2014; Skarlatidis et al., 2015]. This dataset in particular has overlaps where `meet` CE has been initiated and not terminated and then `move` CE is recognized. Thereafter all occurring SDEs during the overlap are irrelevant to `meet` and cannot cause any re-initiation or termination.

Method	TP	FP	FN	Precision	Recall	F_1 score
EC_{crisp}	4008	400	2264	0.9093	0.6390	0.7506
CDA_{HI}	4008	384	2264	0.9125	0.6390	0.7516
CDA_{SI^h}	2437	261	1197	0.9032	0.6706	0.7697
CDA_{SI}	3386	368	2886	0.9019	0.5398	0.6754
$AdaGrad_{HI}$	4008	384	2264	0.9125	0.6390	0.7516
$AdaGrad_{SI^h}$	4077	368	2031	**0.9172**	0.6674	0.7726
$AdaGrad_{SI}$	3148	1542	3124	0.6712	0.5019	0.5743
$MaxMargin_{HI}$	5598	1128	674	0.8323	0.8925	0.8614
$MaxMargin_{SI^h}$	5902	1088	370	0.8443	**0.9410**	**0.8901**
$MaxMargin_{SI}$	3226	1268	1760	0.6599	0.6470	0.6534

TABLE 4.5: Weight learning accuracy of the move CE

Table 4.6 present the running times for learning the weights of the manual definitions. Both training and testing times are averaged across the 10 folds. The results justify the usage of an online weight learner. Both CDA and AdaGrad are an order magnitude faster during training in contrast to the batch max-margin learner. This arises from the fact that batch learning methods require to run inference over the whole dataset in each iteration. CDA is a little faster than AdaGrad in most cases, but it lacks in accuracy and therefore AdaGrad is preferred. Moreover, AdaGrad uses L1-regularization in contrast to CDA which uses L2 and is more appropriate for the task of structure learning, which may introduce a lot of new features and many of which may not be useful. Note that testing times are almost identical because the set of learned clauses used as an input to these methods is the same and only their resulting weights differ. Therefore grounding and inference times do not change substantially. Consequently, the presented experiments justify our choice for selecting AdaGrad in our online structure learning method.

Method	meet		move	
	training	testing	training	testing
CDA_{HI}	1m 28s	11s	2m 27s	17s
CDA_{SI^h}	**1m 24s**	11s	1m 44s	13s
CDA_{SI}	1m 26s	10s	2m 22s	13s
$AdaGrad_{HI}$	1m 40s	11s	**1m 21s**	18s
$AdaGrad_{SI^h}$	1m 26s	11s	1m 35s	15s
$AdaGrad_{SI}$	1m 26s	9s	2m 58s	15s
$MaxMargin_{HI}$	19m 4s	11s	26m 36s	15s
$MaxMargin_{SI^h}$	18m 53s	11s	28m 10s	12s
$MaxMargin_{SI}$	20m 35s	10s	29m 39s	13s

TABLE 4.6: Weight learning running times for meet and move CE

4.4 OSLα Performance

In this section the experiments for our online probabilistic structure learning method are presented. The model is trained natively and its performance is evaluated using 10-fold cross-validation over 19 sequences. The results are compared against the results presented in the previous Section and OSL. In order to achieve the best possible accuracy for OSLα, tuning of the evaluation threshold μ (see Section 3.4) is required. We run structure learning using 10-fold cross validation over 5 distinct values of μ. The results are presented in Figure 4.1. The highest accuracy is achieved by using μ=4 and μ=1 for `meet` and `move` CEs respectively. In the case of `meet` the evaluation threshold smoothly affects accuracy, in contrast to `move` where a lot of important structures are penalized if OSLα becomes more austere and accuracy is decreasing significantly.

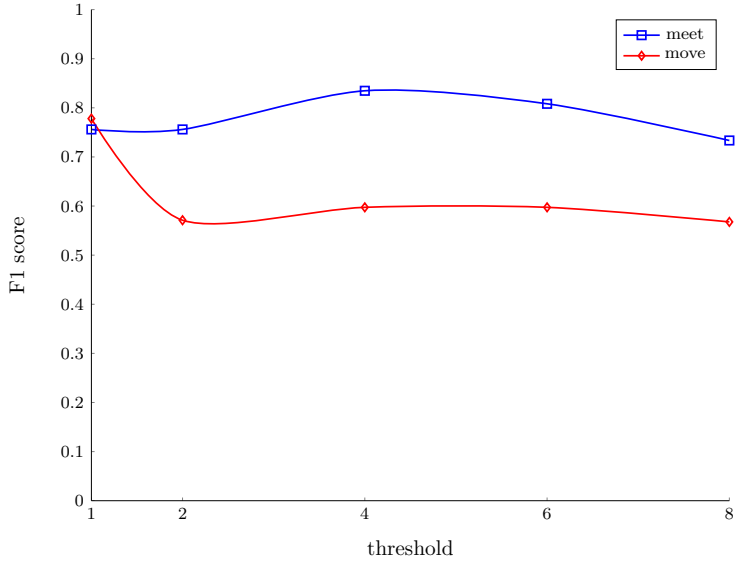

FIGURE 4.1: F1 score over 10 folds for various evaluation threshold values.

Then by using the best thresholds detailed results of OSLα over 10 folds are presented in Tables 4.7(a) and 4.7(b). Note that OSLα can be at least as accurate as AdaGrad training definitions developed by hand or even better (see Tables 4.4 and 4.5 for comparison). Different inertia configurations do not affect the accuracy. That is happening because the dataset contains SDEs in every single timepoint and the definitions learned by OSLα introduce all possible predicates in the bodies of the template atoms (i.e., `InitiatedAt`, `TerminatedAt`) due to the exhaustive hypergraph search. As a consequence, inertia rules are outweighed by the rest of the theory.

Furthermore, Table 4.8 present the running times of OSLα for both CEs. Both training and testing times are averaged across the 10 folds. Training time of `move` is much slower than the one of `meet`. That is because `move` includes another predicate (`OrientationMove`) in its predicate schema and therefore yields more possible structures. Moreover testing times, although pretty fast, still is slower in contrast to the ones presented in the weight

Method	TP	FP	FN	Precision	Recall	F_1 score
$\texttt{EC}_{\texttt{crisp}}$	3099	1413	523	0.6868	0.8556	0.7620
$\texttt{OSL}\alpha_{\texttt{HI}}$	3082	680	540	0.8192	0.8509	0.8347
$\texttt{OSL}\alpha_{\texttt{SI}^\texttt{h}}$	3082	680	540	0.8192	0.8509	0.8347
$\texttt{OSL}\alpha_{\texttt{SI}}$	3082	680	540	0.8192	0.8509	0.8347

(a) Results for the `meet` CE

Method	TP	FP	FN	Precision	Recall	F_1 score
$\texttt{EC}_{\texttt{crisp}}$	4008	400	2264	0.9093	0.6390	0.7506
$\texttt{OSL}\alpha_{\texttt{HI}}$	4718	1138	1554	0.8056	0.7522	0.7780
$\texttt{OSL}\alpha_{\texttt{SI}^\texttt{h}}$	4718	1138	1554	0.8056	0.7522	0.7780
$\texttt{OSL}\alpha_{\texttt{SI}}$	4718	1138	1554	0.8053	0.7522	0.7779

(b) Results for the `move` CE

TABLE 4.7: Results for $\texttt{OSL}\alpha$ for μ= 4 and μ=1 respectively.

Method	`meet`		`move`	
	training	testing	training	testing
$\texttt{OSL}\alpha_{\texttt{HI}}$	**22m 49s**	54s	**1h 56m 2s**	1m 6s
$\texttt{OSL}\alpha_{\texttt{SI}^\texttt{h}}$	23m 4s	49s	1h 59m 6s	59s
$\texttt{OSL}\alpha_{\texttt{SI}}$	23m 17s	55s	1h 58m 50s	1m 2s
OSL	>25h	-	-	-

TABLE 4.8: $\texttt{OSL}\alpha$ running times for `meet` and `move` CE

learning experiments (see Table 4.6). However, structure learning yields a lot of clauses and therefore the grounding of the theory during inference is slower. Then we attempted to perform probabilistic structure learning on this dataset using OSL. Specifically, we began running experiments for the `meet` CE and we terminated the experimentation after 25 hours. During this time OSL had processed only 4 training examples (micro-batches) out of the 17 contained in the first fold. $\texttt{OSL}\alpha$ on the other hand performed 10 fold cross validation for the `meet` CE in about 4 hours.

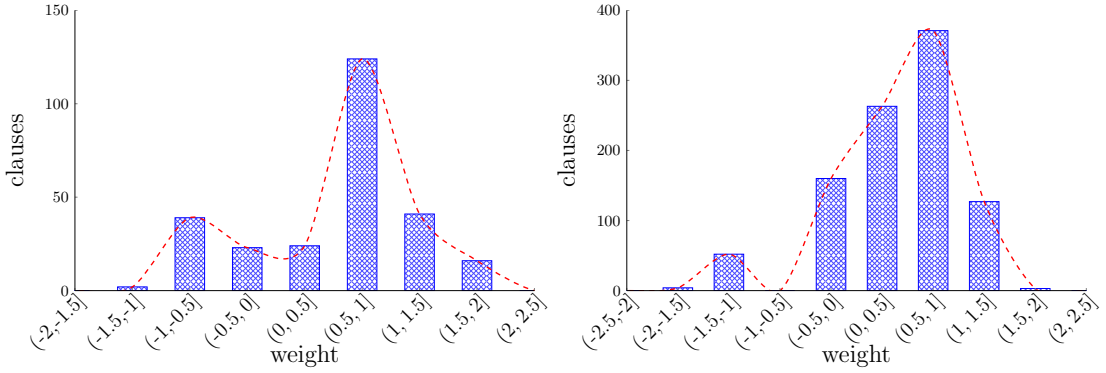

FIGURE 4.2: Weight distibution learned for `meet` (left) and `move` (right)

For effective CE recognition, we prune a portion of the learned weighted structures having weights below a certain threshold ξ (see Section 3.5), for various values of ξ, and present the results in terms of both accuracy and testing time. We begin by running OSLα using all the 19 sequences of the dataset and present a histogram for each CE representing the distribution of weights learned. The distributions are shown in Figure 4.2. The presented histogram give an insight of the portion of the theory that is about to be pruned for each chosen ξ value.

Note that there is a considerable amount of clauses having significantly small weight values. These clauses may be pruned in order to simplify the resulting model without significantly affecting the accuracy, but yielding better inference times. Therefore, we prune the resulting structure for 3 distinct values of ξ and present the change in the behavior with respect to the original model, where all learned structures were retained (ξ=0). The following results are obtained over 10 folds. Figure 4.3 present the reduction in the number of clauses in the resulting theory as ξ increases. This reduction corresponds to missing features that can affect accuracy and on the other hand reduce the inference time.

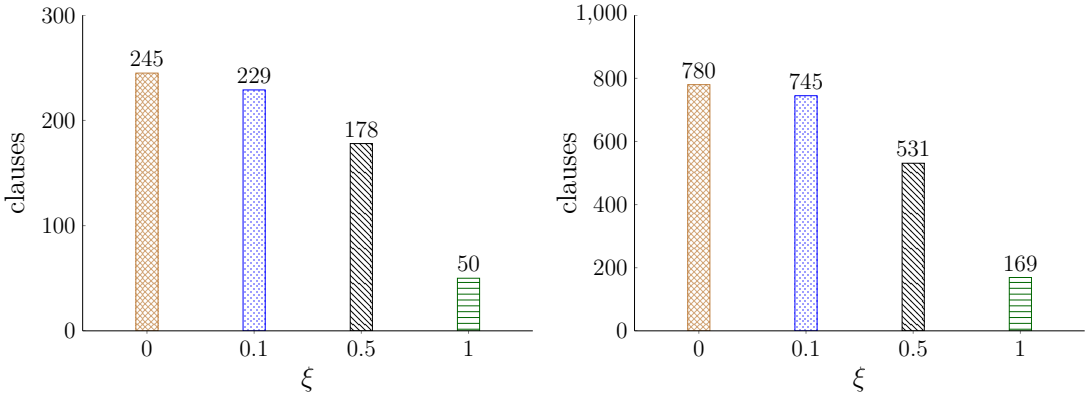

FIGURE 4.3: Reduction in the number of clauses learned as ξ increases for `meet` (left) and `move` (right).

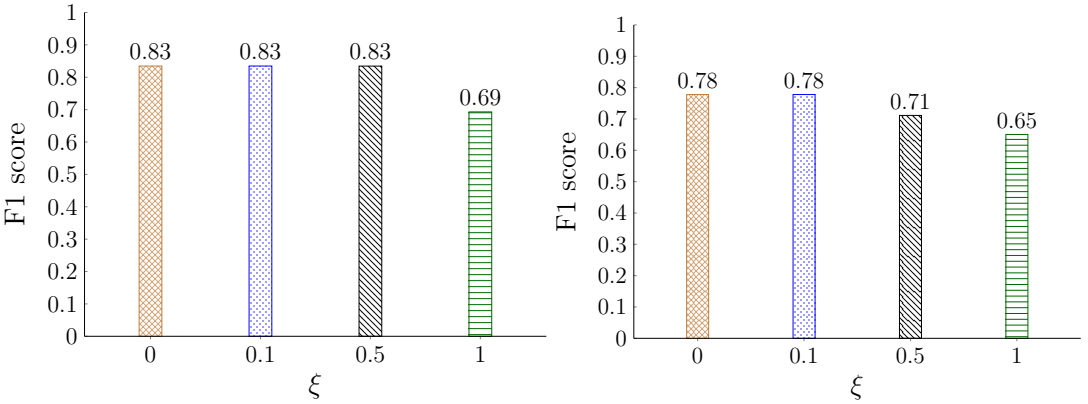

FIGURE 4.4: Effect of F1 score as ξ increases for `meet` (left) and `move` (right).

Figures 4.4 and 4.5 present the effect of ξ in accuracy and testing time. Note that ξ=0.5 results in a slight reduction in accuracy for `move` and no reduction for `meet`, but testing time is much better. Therefore, we can prune a subset of the resulting theory by choosing a value for ξ in order to achieve a minimal set of clauses yielding the best overall accuracy and inference performance.

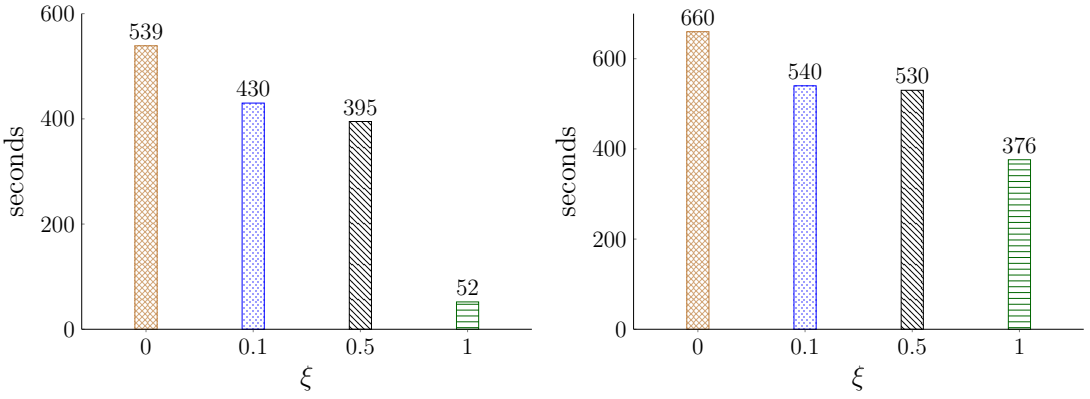

FIGURE 4.5: Reduction of testing time as ξ increases for `meet` (left) and `move` (right).

4.5 Summary

By exploiting of the `MLN–EC` axiomatization, `OSL`α can effectively constrain the space of possible structures and search only for clauses subject to the characteristics introduced by these axioms. By doing so, `OSL`α discovers the weighted structures an order of magnitude faster than OSL. Moreover it can learn definitions equally accurate to the manually developed definitions trained by online weight learning or even outperforming them. Finally, the resulting hypothesis can be pruned leading to faster inference in exchange for a slight reduction of accuracy.

5 Conclusions and Future Work

"Science never solves a problem without creating ten more."
— George Bernard Shaw

In this thesis we focused on probabilistic structure learning under the MLNs framework in order to effectively learn CE definitions for activity recognition. In Section 2.3 we argued that existing structure learning methods cannot effectively handle large training sets or learn complex definitions for temporal reasoning, due to their batch processing nature and greedy strategies. OSL is the first online structure learning approach proposed for MLNs, but it cannot effectively search for clauses in the presence of very large domains of constants or take the background knowledge into account. Moreover none of the existing approaches support first-order logic functions during learning or inference. In order to address these issues we have developed an online structure learning method, extending OSL, that exploits the background knowledge axiomatization in order to effectively constrain the space of possible structures, learn definitions including functions and use an online strategy to handle large training sets.

5.1 Conclusions

In Chapter 3 we presented the OSLα method, in order to address the issue of probabilistic online structure learning by exploiting the background knowledge and learn CE definitions. To demonstrate the features and the performance of the OSLα, we used the probabilistic variant of the Event Calculus (MLN–EC) in the domain of activity recognition. OSLα exploits the axiomatization of MLN–EC, in order to constrain the space of possible structures (i.e., hypergraph) and searches only for clauses having specific characteristics by considering both types of wrongly predicted CEs (false positives and false negatives). The performance of OSLα is demonstrated in Chapter 4 through a series of experiments on a publicly available benchmark dataset. OSLα has been compared against OSL, a

method that uses crisp Event Calculus as well as weight learning alone on manual definitions. The results indicate that OSLα outperforms both OSL and manual definitions trained by the online weight learning methods.

5.2 Future Work

There are several directions in which we would like to extend our work. The main ones are presented below:

Faster Hypergraph Search

The hypergraph search procedure presented in this work effectively constrains the space of possible structures by using the background knowledge but still is an exhaustive search yielding high running times for large domains of constants. Thus, one possible improvement is to employ a heuristic or randomized graph search method like random walks in order to search the hypergraph faster for definitions explaining the data. Moreover, the search procedure can be parallelized in order to get a significant speed up. Specifically, one hypergraph search takes place for each wrongly predicted atom and therefore these searches are independent and can be divided into separated processes running concurrently.

Learn Rules with Negation

Learning definitions that include negated predicates is an important concept of structure learning and gives the learner higher expressive capabilities. Therefore it is desired for the hypergraph search procedure to be able to search for negated literals. As explained in Chapter 3, the hypergraph contains only the true ground atoms as hyperedges and therefore the discovered paths are generalized into positive literals. In order to be able to also discover paths that include negated literals, false ground atoms should exist as hyperedges. Usually false ground atoms are far more than the true ones in a data set, yielding a huge space of hyperedges and making the hypergraph unable to be searched by an exact search procedure like the one OSLα is currently using. Therefore a faster hypergraph search incorporating heuristic methods could possible handle a search space that includes false ground atoms.

Semi-supervised Learning for Handling Missing Data

Investigate the problem of learning the structure of MLNs in the presence of hidden (unobserved) data. Most structure learning approaches (including OSLα) make the

closed world assumption, i.e., whatever is unobserved in the world is considered to be false. Research in the presence of missing data in SRL has mainly focused on weight learning and is based on classical EM. There has also been work on learning the structure of SRL models from hidden data. These approaches, compute the sufficient statistics over the hidden states and perform a greedy hill-climbing search over the clauses.

Bibliography

Apsel, U. and Brafman, R. I. (2012). Lifted MEU by Weighted Model Counting. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press.

Artikis, A., Skarlatidis, A., and Paliouras, G. (2010). Behaviour Recognition from Video Content: a Logic Programming Approach. *International Journal on Artificial Intelligence Tools (JAIT)*, 19(2):193–209.

Biba, M., Ferilli, S., and Esposito, F. (2008). Discriminative Structure Learning of Markov Logic Networks. In *Proceedings of the 18th International Conference on Inductive Logic Programming*, ILP '08, pages 59–76, Berlin, Heidelberg. Springer-Verlag.

Biba, M., Xhafa, F., Esposito, F., and Ferilli, S. (2011). Engineering SLS Algorithms for Statistical Relational Models. In *Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, pages 502–507. IEEE Computer Society.

Blockeel, H. and De Raedt, L. (1998). Top-Down Induction of First-Order Logical Decision Trees. *Artificial Intelligence*, 101(1–2):285–297.

Boros, E. and Hammer, P. L. (2002). Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1–3):155–225.

Bromberg F, Margaritis D, H. V. (2009). Efficient Markov Network Structure Discovery Using Independence Tests. *Journal of Artificial Intelligence Research*, 35(1):449–484.

Casella, G. and George, E. I. (1992). Explaining the Gibbs Sampler. *The American Statistician*, 46(3):167–174.

Collins, M. (2002). Discriminative training methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the ACL Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 10, pages 1–8. Association for Computational Linguistics.

Cowell, R. G., Dawid, A. P., Lauritzen, S. L., and Spiegelhalter, D. J. (2007). *Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks*. Springer Publishing Company, Incorporated, 1st edition.

Damlen, P., Wakefield, J., and Walker, S. (1999). Gibbs sampling for Bayesian non-conjugate and hierarchical models by using auxiliary variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(2):331–344.

Davis, J. and Domingos, P. (2010). Bottom-Up Learning of Markov Network Structure. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 271–278. Omnipress.

De Raedt, L. and Dehaspe, L. (1997). Clausal discovery. *Machine Learning*, 26(2-3):99–146.

den Broeck, G. V., Taghipour, N., Meert, W., Davis, J., and de Raedt, L. (2011). Lifted Probabilistic Inference by First-Order Knowledge Compilation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2178–2185. IJCAI/AAAI.

Dinh, Q. T., Exbrayat, M., and Vrain, C. (2010). Heuristic Method for Discriminative Structure Learning of Markov Logic Networks. In *The Ninth International Conference on Machine Learning and Applications, ICMLA 2010, Washington, DC, USA, 12-14 December 2010*, pages 163–168. IEEE Computer Society.

Domingos, P. and Lowd, D. (2009). *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. (1999). Learning Probabilistic Relational Models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pages 1300–1307, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Gogate, V. and Domingos, P. (2011). Probabilistic Theorem Proving. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 256–265. AUAI Press.

Gonzalez, J., Low, Y., Guestrin, C., and O'Hallaron, D. R. (2009). Distributed Parallel Inference on Large Factor Graphs. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 203–212. AUAI Press.

Hazan, E., Kalai, A., Kale, S., and Agarwal, A. (2006). Logarithmic Regret Algorithms for Online Convex Optimization. In *19th Annual Conference on Learning Theory*, pages 499–513.

Heckerman, D. (1999). Learning in Graphical Models. chapter A Tutorial on Learning with Bayesian Networks, pages 301–354. MIT Press, Cambridge, MA, USA.

Hoos, H. H. and Stützle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann.

Huynh, T. N. and Mooney, R. J. (2008). Discriminative Structure and Parameter Learning for Markov Logic Networks. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 416–423, New York, NY, USA. ACM.

Huynh, T. N. and Mooney, R. J. (2009). Max-Margin Weight Learning for Markov Logic Networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, volume 5781 of *Lecture Notes in Computer Science*, pages 564–579. Springer.

Huynh, T. N. and Mooney, R. J. (2011a). Online Max-Margin Weight Learning for Markov Logic Networks. In *Proceedings of the 11th SIAM International Conference on Data Mining (SDM11)*, pages 642–651, Mesa, Arizona, USA.

Huynh, T. N. and Mooney, R. J. (2011b). Online Structure Learning for Markov Logic Networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD 2011)*, volume 2, pages 81–96.

Joachims, T. (2005). A support vector method for multivariate performance measures. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, pages 377–384. ACM Press.

Joachims, T., Finley, T., and Yu, C.-N. (2009). Cutting-plane training of structural SVMs. *Machine Learning*, 77(1):27–59.

Kautz, H., Selman, B., and Jiang, Y. (1997). A General Stochastic Approach to Solving Problems with Hard and Soft Constraints. In Gu, D., Du, J., and Pardalos, P., editors, *The Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 573–586. AMS.

Kersting, K. (2012). Lifted probabilistic inference. In de Raedt, L., Bessière, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F., and Lucas, P. J. F., editors, *20th European Conference on Artificial Intelligence, Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 33–38. IOS Press.

Kersting, K., Ahmadi, B., and Natarajan, S. (2009). Counting Belief Propagation. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 277–284. AUAI Press.

Khosravi, H., Schulte, O., Man, T., Xu, X., and Bina, B. (2010). Structure Learning for Markov Logic Networks with Many Descriptive Attributes. In Fox, M. and Poole, D., editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press.

Khot, T., Natarajan, S., Kersting, K., and Shavlik, J. (2011). Learning Markov Logic Networks via Functional Gradient Boosting. In *11th IEEE International Conference on Data Mining*, number 1, pages 320–329. IEEE Computer Society.

Khot, T., Natarajan, S., Kersting, K., and Shavlik, J. (2015). Gradient-based Boosting for Statistical Relational Learning: The Markov Logic Network and Missing Data Cases. *Machine Learning*, 100(1):75–100.

Kok, S. and Domingos, P. (2005). Learning the Structure of Markov Logic Networks. In *Proceedings of the 22nd international conference on Machine learning*, pages 441–448. ACM.

Kok, S. and Domingos, P. (2009). Learning Markov Logic Network Structure via Hypergraph Lifting. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 505–512. ACM.

Kok, S. and Domingos, P. (2010). Learning Markov Logic Networks using Structural Motifs. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 551–558. Citeseer.

Kowalski, R. and Sergot, M. (1986). A Logic-based Calculus of Events. *New Generation Computing*, 4(1):67–95.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pages 282–289. Morgan Kaufmann.

Lee, S., Ganapathi, V., and Koller, D. (2006). Efficient structure learning of markov networks using l1 regularization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 817–824.

Lifschitz, V. (1994). Circumscription. In *Handbook of logic in Artificial Intelligence and Logic Programming*, volume 3, pages 297–352. Oxford University Press, Inc.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). Iterated local search. In Glover, F. and Kochenberger, G., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 320–353. Springer US.

Lowd, D. and Davis, J. (2010). Learning Markov Network Structure with Decision Trees. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 334–343, Washington, DC, USA. IEEE Computer Society.

Lowd, D. and Davis, J. (2014). Improving Markov Network Structure Learning Using Decision Trees. *Journal of Machine Learning Research*, 15(1):501–532.

Lowd, D. and Domingos, P. (2007). Efficient Weight Learning for Markov Logic Networks. In *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, volume 4702 of *Lecture Notes in Computer Science*, pages 200–211. Springer.

McCallum, A. (2012). Efficiently inducing features of conditional random fields. *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, abs/1212.2504:2–35.

Mihalkova, L. and Mooney, R. (2007). Bottom-Up Learning of Markov Logic Network Structure. In *Proceedings of the 24th international conference on Machine learning*, pages 625–632. ACM.

Miller, R. and Shanahan, M. (2002). Some Alternative Formulations of the Event Calculus. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, Lecture Notes in Computer Science, pages 452–490. Springer.

Mueller, E. T. (2008). Event Calculus. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 671–708. Elsevier.

Muggleton, S. (1995). Inverse Entailment and Progol. *New Generation Computing*, 13:245–286.

Murphy, K. P. (2002). *Dynamic Bayesian Networks: representation, inference and learning*. PhD thesis, University of California.

Natarajan, S., Khot, T., Kersting, K., Gutmann, B., and Shavlik, J. (2012). Gradient-based Boosting for Statistical Relational Learning: The Relational Dependency Network Case. *Machine Learning*, 86(1):25–56.

Noessner, J., Niepert, M., and Stuckenschmidt, H. (2013). RockIt: Exploiting Parallelism and Symmetry for MAP Inference in Statistical Relational Models. In desJardins, M. and Littman, M. L., editors, *Proceedings of the 27th AAAI Conference on Artificial Intelligence, 2013, USA*. AAAI Press.

Pietra, S. D., Pietra, V. D., and Lafferty, J. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393.

Poon, H. and Domingos, P. (2006). Sound and Efficient Inference with Probabilistic and Deterministic Dependencies. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence*, pages 458–463. AAAI Press.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, 5:239–266.

Rabiner, L. R. and Juang, B.-H. (1986). An introduction to Hidden Markov Models. *Acoustics, Speech, and Signal Processing Magazine (ASSP)*, 3(1):4–16.

Raedt, L. D. (2008). *Logical and Relational Learning: From ILP to MRDM (Cognitive Technologies)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Ravikumar, P., Wainwright, M. J., and Lafferty, J. D. (2010). High dimensional ising model selection using l1 regularized logistic regression. *Ann. Statist.*, 38(3):1287–1319.

Richards, B. L. and Mooney, R. J. (1992). Learning relations by pathfinding. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 50–55. AAAI Press.

Riedel, S. (2008). Improving the Accuracy and Efficiency of MAP Inference for Markov Logic. In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 468–475. AUAI Press.

Shanahan, M. (1999). The Event Calculus Explained. In Wooldridge, M. and Veloso, M., editors, *Artificial Intelligence Today*, volume 1600 of *Lecture Notes in Computer Science*, pages 409–430. Springer.

Singla, P. and Domingos, P. (2005). Discriminative Training of Markov Logic Networks. In *Proceedings of the 20th National Conference on Artificial Intelligence*, pages 868–873. AAAI Press / The MIT Press.

Singla, P. and Domingos, P. (2006). Memory-Efficient Inference in Relational Domains. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence*, pages 488–493. AAAI Press.

Singla, P. and Domingos, P. (2008). Lifted First-Order Belief Propagation. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 1094–1099. AAAI Press.

Skarlatidis, A. (2012). Logical Markov Random Fields (LoMRF): an open-source implementation of Markov Logic Networks.

Skarlatidis, A. (2014). *Event Recognition Under Uncertainty and Incomplete Data*. PhD thesis, Department of Digital Systems, University of Piraeus.

Skarlatidis, A., Paliouras, G., Artikis, A., and Vouros, G. A. (2015). Probabilistic Event Calculus for Event Recognition. *ACM Transactions on Computational Logic*, 16(2):11:1–11:37.

Skarlatidis, A., Paliouras, G., Vouros, G. A., and Artikis, A. (2011). Probabilistic Event Calculus Based on Markov Logic Networks. In *RuleML America*, volume 7018 of *Lecture Notes in Computer Science*, pages 155–170. Springer.

Srinivasan, A. (2004). *The Aleph Manual*.

Sutton, C. and McCallum, A. (2007). An Introduction to Conditional Random Fields for Relational Learning. In Getoor, L. and Taskar, B., editors, *Introduction to Statistical Relational Learning*, pages 93–127. MIT Press.

Taskar, B., Guestrin, C., and Koller, D. (2003). Max-Margin Markov Networks. In *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*, pages 25–32.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large Margin Methods for Structured and Interdependent Output Variables. *Journal of Machine Learning Research*, 6:1453–1484.

Zelle, J. M., Thompson, C. A., Califf, M. E., and Mooney, R. J. (1995). Inducing Logic Programs without Explicit Negative Examples. In *Proceedings of the Fifth International Workshop on Inductive Logic Programming (ILP-95)*, pages 403–416, Leuven, Belgium.