OSLα: Online Structure Learning using Background Knowledge Axiomatization

Evangelos Michelioudakis[1,2], Anastasios Skarlatidis[1], Georgios Paliouras[1], and Alexander Artikis[1,3]

Institute of Informatics and Telecommunications, NCSR "Demokritos" [1]
School of Electronic and Computer Engineering, Technical University of Crete[2]
Department of Maritime Studies, University of Piraeus[3]
{vagmcs,anskarl,paliourg,a.artikis}@iit.demokritos.gr

Abstract. We present OSLα — an online structure learner for Markov Logic Networks (MLNs) that exploits background knowledge axiomatization in order to constrain the space of possible structures. Many domains of interest are characterized by uncertainty and complex relational structure. MLNs is a state-of-the-art Statistical Relational Learning framework that can naturally be applied to domains governed by these characteristics. Learning MLNs from data is challenging, as their relational structure increases the complexity of the learning process. In addition, due to the dynamic nature of many real-world applications, it is desirable to incrementally learn or revise the model's structure and parameters. Experimental results are presented in activity recognition using a probabilistic variant of the Event Calculus (MLN-EC) as background knowledge and a benchmark dataset for video surveillance.

Keywords: Markov Logic Networks, Event Calculus, Uncertainty

1 Introduction

Many real-world application domains are characterized by both uncertainty and complex relational structure. Regularities in these domains are very hard to identify manually, and thus automatically learning them from data is desirable. The field of Statistical Relational Learning (SRL) [7] concerns the induction of probabilistic knowledge by combining the powers of logic and probability. One of the logic-based frameworks that handles uncertainty, proposed in the area of SRL, is Markov Logic Networks (MLNs) [24] which combines first-order logic and probabilistic graphical models.

Structure learning approaches that focus on MLNs have been successfully applied to a variety of applications where uncertainty holds [6]. However, most of these methods are batch algorithms that cannot handle large training sets or large data streams as they are bound to repeatedly perform inference over the entire training set in each learning iteration. This is computationally expensive, rendering these algorithms inapplicable to real-world applications. Huynh and Mooney [12] proposed an online strategy, called OSL, for updating both the

structure and the parameters of the model, in order to effectively handle large training datasets. Nevertheless, OSL does not exploit background knowledge during the search procedure and explores structures that are very common and therefore largely useless for the purposes of learning, yielding models that are not adequate generalizations of the data.

We propose the OSLα online structure learner for MLNs, which extends OSL by exploiting a given background knowledge, in order to effectively constrain the search space of possible structures during learning. The space is constrained subject to characteristics imposed by the rules governing a specific task, herein stated as axioms. To demonstrate the benefits of OSLα we focus on the domain of activity recognition. As a background knowledge we are employing MLN–EC [27], a probabilistic variant of the Event Calculus [20] for event recognition applications.

Running Example. In activity recognition the goal is to recognize *composite events* (CE) of interest given an input stream of *simple derived events* (SDEs). CEs can be defined as relational structures over sub-events, either CEs or SDEs, and capture the knowledge of a target application. Due to the dynamic nature of 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 curation of event definitions is a tedious and cumbersome process and thus machine learning techniques to automatically derive the definitions are essential. The proposed OSLα method is tested on the task of activity recognition from surveillance video footage. The goal is to recognize activities that take place between multiple persons, e.g. people meeting and moving together, by exploiting information about observed activities of individuals. The input stream of SDEs, represents people walking, running, staying active, or inactive, and spatial relations, e.g. persons being relatively close to each other.

The remainder of the paper is organized as follows. Section 2 provides background on MLNs and MLN–EC. Section 3 discusses related work on structure learning. Section 4 describes our proposed method for online structure learning. Section 5 reports the experimental results and Section 6 proposes directions for future work and concludes.

2 Background

2.1 Markov Logic Networks

Markov Logic Networks (MLNs) [24] consist of weighted first-order formulas. They provide a way of softening the constraints that are imposed by the formulas and facilitate probabilistic inference. Hence, unlike classical logic, all worlds in MLNs are possible and they are quantified by a certain probability. In event recognition the focus is on discriminative MLNs [26]. Let X be a set of evidence atoms, and Y a set of query atoms. The former correspond to the input SDEs while the latter correspond to the CEs of interest in event recognition. Then the conditional probability of $\mathbf{y}$ given $\mathbf{x}$ is defined as follows:

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

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

2.2 MLN–EC: Probabilistic Event Calculus based on MLNs

MLN–EC [27] is a probabilistic variant of the discrete Event Calculus [20] in MLNs for event recognition applications. The ontology of MLN–EC consists of *time-points*, *events* and *fluents*, represented by the finite sets $\mathcal{T}$, $\mathcal{E}$ and $\mathcal{F}$, respectively. The underlying time model is linear and represented by integers. A *fluent* is a property whose value may change over time by the occurrence of a particular *event*. MLN–EC comprises the core domain-independent axioms of Event Calculus defining whether a fluent holds or not at a specific time-point. In addition, the domain-independent axiomatization incorporates the common sense *law of inertia*, according to which fluents persist over time, unless they are affected by an event occurrence. MLN–EC axioms (1) and (2), shown below, determine when a fluent holds and axioms (3) and (4) when a fluent does not hold. Variables and functions start with a lower-case letter and are assumed to be universally quantified. Predicates start with an upper-case letter and predicate `Next` expresses successive time-points to avoid numerical calculations.

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

$$\texttt{HoldsAt}(f, t{+}1) \Leftarrow \texttt{HoldsAt}(f, t) \wedge \neg\texttt{TerminatedAt}(f, t) \wedge \texttt{Next}(t, t{+}1) \quad (2)$$

$$\neg\texttt{HoldsAt}(f, t{+}1) \Leftarrow \texttt{TerminatedAt}(f, t) \wedge \texttt{Next}(t, t{+}1) \quad (3)$$

$$\neg\texttt{HoldsAt}(f, t{+}1) \Leftarrow \neg\texttt{HoldsAt}(f, t) \wedge \neg\texttt{InitiatedAt}(f, t) \wedge \texttt{Next}(t, t{+}1) \quad (4)$$

MLN–EC combines composite event definitions with the domain-independent axioms of MLN–EC (1)–(4), generating a compact knowledge base that serves as a pattern for the production of Markov Networks, and enables probabilistic inference and machine learning. The compact knowledge base is generated by performing predicate completion [20] – a syntactic transformation that translates formulas into logically stronger ones. The aim of predicate completion is to rule out all conditions which are not explicitly entailed by the given formulas and thus to introduce closed-world assumption to first-order logic.

3 Related Work

Learning the MLN structure is a task that has received much attention lately. The main approaches to this task stem either from graphical models [22, 8, 18] or Inductive Logic Programming (ILP) [23, 4]. Since MLNs represent probability distributions, better results are obtained by evaluation functions based on likelihood, rather than typical ILP ones like accuracy and coverage [14].

Several approaches have been proposed to date [19, 9, 2, 17, 15, 16, 13], using various strategies to search the space of possible structures. Most of these approaches are batch learning algorithms that cannot handle very large training sets, due to their requirement to load all data in memory and carry out inference in each iteration. Moreover, most of these algorithms are strictly data-driven and thus they only seek to improve the likelihood of known true worlds.

Huynh and Mooney [12] proposed OSL that updates both the structure and the parameters of the model using an incremental approach whereby training data are consumed in (non-overlapping) micro-batches. Using incorrect predictions of the current model, OSL searches for clauses, using relational pathfinding over a hypergraph [25] constrained by mode declarations [21], and estimates or updates their parameters using the AdaGrad learner [5]. The hypergraph may be seen as a representation of the search space that contains true ground predicates, while the paths found during the mode-guided search may be seen as conjunctions of ground predicates, that are eventually generalized to clauses.

OSL does not exploit background knowledge that may be provided to constrain the search space and typically explores many structures (paths) that are not useful. Specifically, even by 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 and 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 and use only meaningful subspaces that may be found by exploiting the background knowledge axiomatization.

Finally, all aforementioned approaches assume that domains do not contain functions, which are useful in several applications, such as activity recognition.

4 Online Structure Learning using Background Knowledge Axiomatization

Figure 1 presents the components of OSLα. The background knowledge consists of the MLN–EC axioms (i.e., domain-independent rules) and an already known (possibly empty) hypothesis (i.e., set of clauses). At any step t of the online procedure a training example (micro-batch) $\boldsymbol{\mathcal{D}}_t$ arrives containing simple derived events (SDEs), e.g. two persons walking individually, their distance being less than 34 pixel positions and having the same orientation. Then, $\boldsymbol{\mathcal{D}}_t$ is used together with the already learnt hypothesis to predict the truth values y_t^P of the

composite events (CEs) of interest. This is achieved by (maximum a posteriori) MAP inference based on LP-relaxed Integer Linear Programming [10]. Given $\boldsymbol{\mathcal{D}}_t$ OSLα constructs a hypergraph that represents the space of possible structures as graph paths. Then for all incorrectly predicted CEs the hypergraph is searched (guided by MLN–EC axioms) for definite clauses explaining these CEs. The paths discovered during the search are translated into clauses and evaluated. The resulting set of retained clauses is used for weight learning. Finally, the set of weighted clauses is appended to the hypothesis $\boldsymbol{\mathcal{H}}_t$ and the whole procedure is repeated for the next training example $\boldsymbol{\mathcal{D}}_{t+1}$.

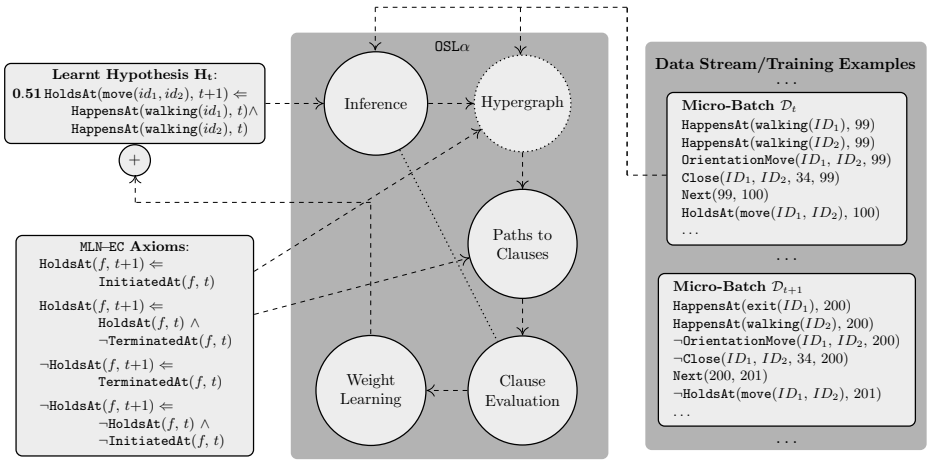

Fig. 1: The procedure of OSLα.

4.1 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 already known hypothesis $\boldsymbol{\mathcal{H}}$ (herein CE definitions). Each axiom $\alpha \in \mathcal{A}$ must not contain any free variables, meaning variables only appearing in a single predicate. It should also contain exactly one so-called *template predicate* and at least one *query predicate*. In the case of MLN–EC, $\mathcal{A}$ contains the four axioms (1)–(4), 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. They form the target CE patterns that we want to learn.

MLN–EC axioms can be used as a template $\mathbf{T}$ over all possible structures in order to search only for explanations of the template predicates. Upon doing so, OSLα does not need to search over time sequences, instead only needs to find appropriate bodies over the current time-point for the following definite clauses:

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

The body of these definitions is a conjunction of n literals $\ell_1 \wedge \cdots \wedge \ell_n$, which can be seen as a hypergraph path, as we shall explain in the following sections.

Given the set of axioms $\mathcal{A}$, OSLα partitions it into templates. Each template $\mathbf{T}_i$ contains axioms with identical Cartesian product of domain types over their template predicate variables. MLN–EC axioms (1)–(4) should all belong to one template $\mathbf{T}_1$ because InitiatedAt and TerminatedAt both have joint domain $\mathcal{F} \times \mathcal{T}$. The resulting template $\mathbf{T}_1$ is used during relational pathfinding (see Section 4.3) to find an initial search set $\mathcal{I}$ of ground template predicates and search the space of possible structures for specific bodies of the definite clauses. A template $\mathbf{T}_i$ essentially provides mappings of its axioms to the template predicates that appear in the bodies of these axioms. For instance, axiom (1) of $\mathbf{T}_1$ will be mapped to the predicate $\texttt{InitiatedAt}(f, t)$ since the aim is to construct a rule for this template predicate.

4.2 Hypergraph and Relational Pathfinding

Similar to OSL, at each step t OSLα receives an example $\mathbf{x}_t$, representing the evidence part of $\boldsymbol{\mathcal{D}}_t$ and produces the predicted label $\mathbf{y}_t^P = \text{argmax}_{\mathbf{y} \in \boldsymbol{\mathcal{Y}}} \langle \mathbf{w}, \mathbf{n}(\mathbf{x}_t, \mathbf{y}) \rangle$ using MAP inference. It then receives the true label $\mathbf{y}_t$ and finds all ground atoms that are in $\mathbf{y}_t$ but not in $\mathbf{y}_t^P$ denoted as $\Delta y_t = \mathbf{y}_t \setminus \mathbf{y}_t^P$. Hence, Δy_t contains the false positives/negatives of the inference step. In contrast to OSL, OSLα considers all misclassified (false positives/negatives) ground atoms instead of just the true ones (false negatives) in order to find InitiatedAt definitions that correct the false negatives and respectively TerminatedAt for the false positives. OSLα searches the ground-truth world ($\mathbf{x}_t$,$\mathbf{y}_t$) for clauses specific to the axioms defined in the background knowledge using the constructed templates $\mathbf{T}_i$.

In order to discover useful clauses specific to the set of incorrectly predicted atoms Δy_t, OSLα uses relational pathfinding [25]. It considers $\boldsymbol{\mathcal{D}}_t$ as a hypergraph having constants as nodes and true ground atoms as hyperedges that connect the nodes appearing as its arguments. Hyperedges are a generalization of edges connecting any number of nodes. OSLα searches the hypergraph for paths that connect the arguments of an input incorrectly predicted atom. Functions present in $\boldsymbol{\mathcal{D}}_t$ are transformed into auxiliary predicates (with the prefix AUX) that model the behavior of a function and are required to indirectly include functions in the hypergraph. For example the predicate AUXwalking matches the return values of the function walking and has arity increased by 1 in order to incorporate the return type of the function as an argument of the auxiliary predicate.

A hypergraph representing the training example $\boldsymbol{\mathcal{D}}_t$ of Figure 1 is presented on the left of Figure 2. For each incorrectly predicted ground atom in Δy_t (herein incorrectly predicted CEs), relational pathfinding searches for all paths up to a predefined length l. A path of hyperedges corresponds to a conjunction of

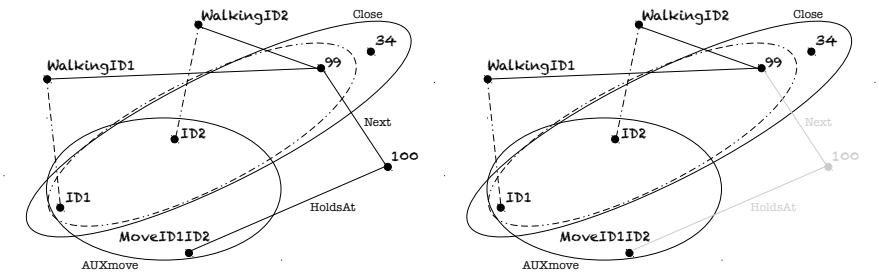

Fig. 2: Initial hypergraph (left) and reduced hypergraph (right). Unlabelled continuous lines represent `HappensAt` predicates, while unlabelled dashed lines and dashed ellipses represent `AUXwalking` and `OrientationMove` respectively.

true ground atoms connected by their arguments and can be generalized into a conjunction of variabilized literals. For example consider that the predicted label y_t^P says that $\texttt{HoldsAt}(\mathit{MoveID_1}\,\mathit{ID_2}, 100)$ is false, while supervision in $\boldsymbol{\mathcal{D}}_t$, that is y_t, says that it is true. Therefore it is an incorrectly predicted atom and the hypergraph should be searched for paths explaining the misclassified CE. Below, we present two of the paths that can be found by searching the left hypergraph of Figure 2 for paths up to length $l = 7$.

$$\big\{\texttt{HoldsAt}(\mathit{MoveID_1}\,\mathit{ID_2}, 100), \texttt{Next}(\mathit{99}, \mathit{100}), \texttt{HappensAt}(\mathit{WalkingID_1}, 99), \texttt{HappensAt}(\mathit{WalkingID_2}, 99), \texttt{AUXwalking}(\mathit{WalkingID_1}, \mathit{ID_1}), \texttt{AUXwalking}(\mathit{WalkingID_2}, \mathit{ID_2}), \texttt{AUXmove}(\mathit{MoveID_1ID_2}, \mathit{ID_1}, \mathit{ID_2})\big\} \tag{5}$$

$$\big\{\texttt{HoldsAt}(\mathit{MoveID_1}\,\mathit{ID_2}, 100), \texttt{Next}(\mathit{99}, \mathit{100}), \texttt{Close}(\mathit{ID_1}, \mathit{ID_2}, 34, 99), \texttt{AUXmove}(\mathit{MoveID_1ID_2}, \mathit{ID_1}, \mathit{ID_2})\big\} \tag{6}$$

Similar to [12], in order to speed up relational pathfinding, $\texttt{OSL}\alpha$ uses path mode declarations as language bias to constrain the space of paths. A $\texttt{modep}(r, p)$ has two components: a recall number $r \in \mathbb{N}_0$, and an atom p whose arguments are place-markers optionally preceded by symbol '#'. A place-marker is '+' (input), '−' (output), or '.' (ignore). The symbol '#' preceding place-markers specifies that this particular predicate argument will remain constant after the generalization of the path. 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. We also introduce mode declarations for functions, defined as $\texttt{modef}(r, p)$, that are used to constrain auxiliary predicates in the hypergraph.

The hypergraph is constructed from a training example $\boldsymbol{\mathcal{D}}_t$, by only adding true ground atoms in $\boldsymbol{\mathcal{D}}_t$ that are input or 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. Template predicates are not added in the hypergraph because they are not allowed to appear in the body of the definite clause. Hence, OSLα does not support recursive definitions.

4.3 Template Guided Search

Starting from each incorrectly predicted ground atom in Δy_t, we use the templates $\mathbf{T}_i$ constructed at the initial steps of the algorithm in order to find the corresponding ground template predicates for which the axioms belonging in $\mathbf{T}_i$ are satisfied by the current training example. As stated in Section 4.1 there is only one template $\mathbf{T}_1$ containing all the axioms of MLN–EC. OSLα considers each axiom $\alpha \in \mathbf{T_1}$ in turn. Assume, for example, that one of these is axiom (1) and we have incorrectly predicted that the ground atom $\texttt{HoldsAt}(CE, T_4)$ is false (false negative). We substitute the constants of $\texttt{HoldsAt}(CE, T_4)$ into axiom (1). The result of the substitution will be the following partially ground axiom:

$$\texttt{HoldsAt}(CE, T_4) \Leftarrow \texttt{Next}(t, T_4) \wedge \texttt{InitiatedAt}(CE, t) \tag{7}$$

If after the substitution there are no variables left in the template predicate of the axiom, OSLα adds the ground template predicate to the initial search set $\mathcal{I}$, containing all ground template predicates, and moves to the next axiom in the template $\mathbf{T}_1$. In case there are variables left, such as in axiom (7) were InitiatedAt has one remaining variable t, OSLα searches for all literals in the axiom sharing variables with the template predicate. Here the only literal sharing the remaining variable t is Next. For those literals, it searches the training data for all jointly ground instantiations among those satisfying the axiom. Because t represents time-points and Next describes successive time-points, there will be only one true grounding of Next in the training data having as argument the constant T_3. OSLα substitutes the constant T_3 into axiom (7) and adds $\texttt{InitiatedAt}(CE, T_3)$ to the initial search set $\mathcal{I}$. The same applies for axioms (3) and (4) determining the termination conditions in the case of a false positive.

For each ground template predicate in the resulting initiation set $\mathcal{I}$, the mode-guided relational pathfinding is used to search the hypergraph for an appropriate body. It recursively adds to the path hyperedges (i.e., ground atoms) that satisfy the mode declarations. The search terminates when the path reaches a specified maximum length or when no new hyperedges can be added.

By employing this procedure, the hypergraph is essentially reduced to contain only ground atoms explaining the template predicates. Consider the hypergraph presented on the left of Figure 2. By exploiting the Event Calculus axioms, the hypergraph is reduced to contain only predicates that explain the InitiatedAt and TerminatedAt predicates as presented in the right of Figure 2. The paths (5) and (6) are pruned by removing the Next and HoldsAt predicates, resulting into the paths (8) and (9) shown below. The pruning resulting from the template guided search is essential to learn Event Calculus definitions, because the size of the search space becomes independent of time.

$$\{\texttt{InitiatedAt}(MoveID_1\,ID_2,\,99),\ \texttt{HappensAt}(WalkingID_1,\,99),\ \texttt{HappensAt}(WalkingID_2,\,99),\ \texttt{AUXwalking}(WalkingID_1, ID_1),\ \texttt{AUXwalking}(WalkingID_2, ID_2),\ \texttt{AUXmove}(MoveID_1ID_2, ID_1, ID_2)\} \tag{8}$$

$$\{\texttt{InitiatedAt}(MoveID_1\,ID_2,\,99),\ \texttt{Close}(ID_1,\,ID_2,\,34,\,99),\ \texttt{AUXmove}(MoveID_1ID_2, ID_1, ID_2)\} \tag{9}$$

4.4 Clause Creation and Evaluation

In order to generalize paths into first-order clauses, we replace each constant k_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 as head the template predicate present in each path. The auxiliary predicates are converted back into functions. Therefore, from the paths (8) and (9), the following definite clauses will be created:

$$\texttt{InitiatedAt}(\texttt{move}(id_1, id_2),\,t) \Leftarrow \texttt{HappensAt}(\texttt{walking}(id_1),\,t) \wedge \texttt{HappensAt}(\texttt{walking}(id_2),\,t) \tag{10}$$

$$\texttt{InitiatedAt}(\texttt{move}(id_1, id_2),\,t) \Leftarrow \texttt{Close}(id_1,\,id_2,\,34,\,t) \tag{11}$$

According to the definitions (10) and (11), the `move` CE is initiated either when both entities are walking or the distance between them is less than 34 pixel positions. These definite clauses can be used together with the axioms of the background knowledge in order to eliminate all template predicates by exploiting equivalences resulting from predicate completion.

After the elimination process all resulting formulas are converted into clausal normal form (CNF). Therefore the resulting set of clauses is independent of the template predicates. Evaluation takes place for each clause c individually. The difference between the number of true groundings of c in the ground-truth world $(\mathbf{x}_t, \mathbf{y}_t)$ and those in predicted world $(\mathbf{x}_t, \mathbf{y}_t^P)$ is then computed (note that $\mathbf{y}_t^P$ was predicted without c). Only clauses whose difference in the number of groundings is greater than or equal to a predefined threshold μ will be added to the MLN:

$$\Delta n_c = n_c(\mathbf{x}_t, \mathbf{y}_t) - n_c(\mathbf{x}_t, \mathbf{y}_t^P) \geq \mu \tag{12}$$

The intuition behind this measure is to add to the hypothesis $\boldsymbol{\mathcal{H}}$ clauses whose coverage of the ground-truth world is significantly (according to μ) greater than that of 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 this transformation may change 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 (10)–(11). After predicate completion, these rules will be replaced by the following formula:

$$\begin{aligned}&\texttt{InitiatedAt}(\texttt{move}(id_1, id_2), t) \Leftrightarrow \\ &\quad \big(\texttt{HappensAt}(\texttt{walking}(id_1), t) \wedge \texttt{HappensAt}(\texttt{walking}(id_2), t)\big) \bigvee \\ &\quad \texttt{Close}(id_1, id_2, 34, t)\end{aligned} \tag{13}$$

The resulting rule (13) defines all conditions under which the `move` CE is initiated. Based on the equivalence in formula (13), the domain-independent axiom (1) of `MLN-EC` automatically produces the following free of template predicates (i.e, `InitiatedAt`, `TerminatedAt`) rules:

$$\begin{aligned}&\texttt{HoldsAt}(\texttt{move}(id_1, id_2), t{+}1) \Leftarrow \\ &\quad \texttt{HappensAt}(\texttt{walking}(id_1), t) \wedge \\ &\quad \texttt{HappensAt}(\texttt{walking}(id_2), t)\end{aligned} \tag{14}$$

$$\begin{aligned}&\texttt{HoldsAt}(\texttt{move}(id_1, id_2), t{+}1) \Leftarrow \\ &\quad \texttt{Close}(id_1, id_2, 34, t)\end{aligned} \tag{15}$$

Similarly, the inertia axiom (4) produces:

$$\begin{aligned}&\neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2), t{+}1) \Leftarrow \\ &\quad \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2), t) \wedge \\ &\quad \neg\big((\texttt{HappensAt}(\texttt{walking}(id_1), t) \wedge \texttt{HappensAt}(\texttt{walking}(id_2), t)) \bigvee \\ &\qquad \texttt{Close}(id_1, id_2, 34, t)\big)\end{aligned} \tag{16}$$

Consider now, that during the evaluation process the definite clause (15) yields a score less than μ and therefore must be discarded. Then, the resulting hypothesis is reduced to rule (14) produced by axiom (1), as well as rule (17) produced by axiom (4) presented below:

$$\begin{aligned}&\neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2), t{+}1) \Leftarrow \\ &\quad \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2), t) \wedge \\ &\quad \neg\big(\texttt{HappensAt}(\texttt{walking}(id_1), t) \wedge \texttt{HappensAt}(\texttt{walking}(id_2), t)\big)\end{aligned} \tag{17}$$

4.5 Weight Learning

The weights of all retained clauses are optimized by the AdaGrad online learner [5]. At each step t of `OSL`α the learnt hypothesis may be updated by adding new clauses found during the hypergraph search and therefore the resulting set of clauses $\mathcal{C}_t$ may be different from the set $\mathcal{C}_{t-1}$. In order for AdaGrad to be able to apply weight updates to a constantly changing theory, `OSL`α searches for clauses in the current theory $\mathcal{C}_t$ that are θ-subsumed [3] by a clause in the previous theory, in order to inherit its weight. This way the already learnt weight values are transferred to the next step of the procedure. All other clauses are considered new and their weights are set to an initial value close to zero. To illustrate the procedure consider a set of definite clauses $\mathcal{C}_{t-1}$ learnt at step $t{-}1$, including rules (10) as well as rule (18) presented below:

$$\begin{aligned}&\texttt{TerminatedAt}(\texttt{move}(id_1, id_2), t) \Leftarrow \\ &\quad \texttt{HappensAt}(\texttt{inactive}(id_1), t) \wedge \\ &\quad \texttt{HappensAt}(\texttt{active}(id_2), t)\end{aligned} \tag{18}$$

By performing predicate completion upon the set $\mathcal{C}_{t-1}$ and using the MLN–EC axioms to eliminate the template predicates, the following hypothesis arises:

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

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

The set Σ_{t-1} contains specialized definitions of axioms (1) and (3), specifying that a fluent holds (or does not hold) when its initiation (or termination) conditions are met. The set Σ'_{t-1} contains specialized definitions of the inertia axioms (2) and (4), determining whether a specific fluent continues to hold or not at any instance of time. Weights for both sets are estimated. In the next learning step t of OSLα the set of definite clauses $\mathcal{C}_t$ may be expanded by the following learnt definite clause:

$$\begin{aligned} \texttt{TerminatedAt}(\texttt{move}(id_1, id_2), t) \Leftarrow \\ \texttt{HappensAt}(\texttt{exit}(id_1), t) \end{aligned} \tag{19}$$

Similarly to $\mathcal{C}_{t-1}$, by applying predicate completion to $\mathcal{C}_t$ and eliminating the template predicates using the MLN–EC axioms, a different hypothesis arises. Σ_t includes the rules of Σ_{t-1}, as well as the following, resulting from rule (19):

$$\begin{aligned} \neg\texttt{HoldsAt}(\texttt{move}(id_1, id_2), t{+}1) \Leftarrow \\ \texttt{HappensAt}(\texttt{exit}(id_1), t) \end{aligned}$$

Σ'_t includes the first rule appearing in Σ'_{t-1}, as well as the following rule:

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

Note that in the set Σ_t a new rule has appeared and in the set Σ'_t the second rule changed by incorporating a new 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. Using θ-subsumption, OSLα searches for clauses in $\mathcal{C}_t$ that are subsumed by clauses in $\mathcal{C}_{t-1}$ to inherit their weights. In the example above, the first rule of Σ_t and Σ'_t, as well as the second rule of Σ_t are identical to the previous ones.

Moreover, the second rule of Σ'_t is θ-subsumed by the second rule of Σ'_{t-1}. Hence the weights of the old rules will be used for the new ones. The last rule of Σ_t is completely new and its weight is set to a default initial value.

At the end of the OSLα learning we can choose to remove clauses whose weights are smaller than a predefined threshold ξ. Hence, the hypothesis may be pruned significantly, with negligible penalty in accuracy.

All algorithms composing OSLα (e.g., hypergraph construction), in pseudo-code, are available from `iit.demokritos.gr/~vagmcs/pub/osla/appendix.pdf`.

5 Empirical Evaluation

We evaluate OSLα in activity recognition, using the publicly available benchmark dataset of the CAVIAR project[1]. The dataset comprises 28 surveillance videos, where each frame is annotated by human experts from the CAVIAR team on two levels. The first level contains SDEs that concern activities of individual persons or the state of objects. The second level contains CE annotations, describing the activities between multiple persons and/or objects, i.e., people meeting and moving together, leaving an object and fighting.

5.1 Experimental Setup

The input to the learning methods being compared is a stream of SDEs along with the CE annotations. The SDEs represent people walking, running, staying active, or inactive. The first and last time that a person is tracked is represented by the enter and exit SDEs. Additionally, the coordinates of tracked persons are also used to express qualitative spatial relations, e.g. two persons being relatively close to each other. The CE supervision indicates when each of the CEs holds. The structure of the training sequences is presented Figure 1. Each sequence is composed of input SDEs (`HappensAt`), precomputed spatial constraints (`Close`), and the corresponding CE annotations (`HoldsAt`). Negated predicates in the sequence state that the truth value of the corresponding predicate is False.

From the 28 videos, 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 CEs. 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 63147 SDEs and 25738 annotated CE instances. There are 6272 example instances in which `move` occurs and 3722 in which `meet` occurs. 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`.

Throughout the experimental analysis, the evaluation results were obtained using MAP inference, as per [10] and are presented in terms of True Positives

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

(TP), False Positives (FP), False Negatives (FN), Precision, Recall and F_1 score. All reported statistics are micro-averaged over the instances of recognized CEs using 10-fold cross validation over the 19 sequences. The average SDEs per fold are 56832 and the average positive CEs are 3350 and 5600 for `meet` and `move` respectively. The experiments were performed in a computer with an Intel i7 4790@3.6GHz processor (4 cores and 8 threads) and 16GiB of RAM, running Apple OSX version 10.11. All weight and structure learning methods are implemented in LoMRF[2], an open-source implementation of MLNs.

We ran experiments using the AdaGrad [5] and CDA [11] online weight learners as well as a batch max-margin learner [10], using manual definitions developed in [1][3]. 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 `move` starts being recognized. Similarly, when two persons walk away from each other, then `move` stops being recognized. We also include in the experiments the results of the logic-based activity recognition method of [1], hereafter $\texttt{EC}_{\texttt{crisp}}$, that employs a different variant of the Event Calculus, uses the same manual definitions of CEs and cannot perform probabilistic reasoning.

Method	Precision	Recall	$\mathbf{F_1}$ score
$\texttt{EC}_{\texttt{crisp}}$	0.6868	0.8556	0.7620
`MaxMargin`	**0.9189**	0.8133	**0.8629**
`CDA`	0.9061	0.4878	0.6342
`AdaGrad`	0.7228	**0.8547**	0.7833
`OSL`α	0.8192	0.8509	0.8347

(a) Results for the `meet` CE ($\mu = 4$)

Method	Precision	Recall	$\mathbf{F_1}$ score
$\texttt{EC}_{\texttt{crisp}}$	0.9093	0.6390	0.7506
`MaxMargin`	0.8443	**0.9410**	**0.8901**
`CDA`	0.9032	0.6706	0.7697
`AdaGrad`	**0.9172**	0.6674	0.7726
`OSL`α	0.8056	0.7522	0.7780

(b) Results for the `move` CE ($\mu = 1$))

Table 1: Recognition accuracy for the two CEs.

Method	`meet`	`move`
`OSL`α	00h 23m 04s	1h 59m 06s
OSL	> 25h 00m 00s	-

Table 2: Average training times for `meet` and `move` CE.

[2] `https://github.com/anskarl/LoMRF`

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

5.2 Experimental Results

We ran structure learning using 10-fold cross validation over 5 distinct values of the evaluation threshold μ — see formula (12). (All other numerical thresholds were manually set.) The highest accuracy is achieved by using μ=4 and μ=1 for the `meet` and `move` CEs respectively. See Tables 1a and 1b. The batch max-margin weight learning yields the best overall accuracy due the fact that it uses all the data at once to estimate the weights. AdaGrad is the second best choice among the weight learners as it yields more accurate results as opposed to CDA. It also outperforms the unweighted manual knowledge base $\texttt{EC}_{\texttt{crisp}}$. `OSL`$\alpha$ achieves very good results, outperforming AdaGrad in the `meet` CE and achieving a similar F_1 score with it in the `move` CE. This is very encouraging given that `OSL`α does not use manually curated rules.

Table 2 presents the averaged training times for the two CEs. The training time for `move` is much higher than that for `meet`. This is because `move` includes the predicate `OrientationMove` in its predicate mode declarations, leading to a larger search space. We also 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 of the first fold. `OSL`α on the other hand performed 10 fold cross validation for the `meet` CE in about 4 hours.

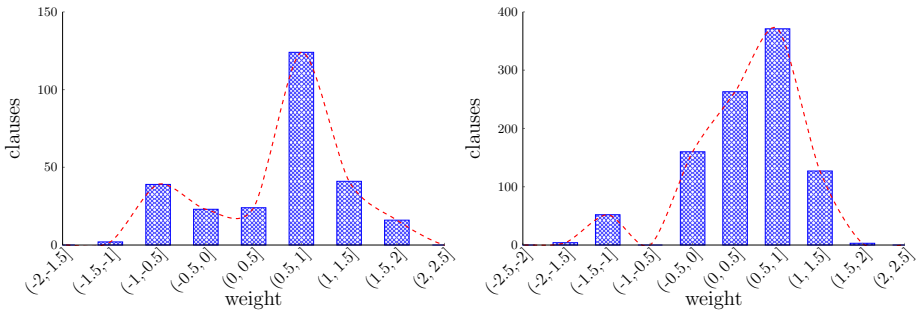

Fig. 3: Weight distribution learned for `meet` (left) and `move` (right).

In order to secure efficient CE recognition, we prune a portion of the learned weighted structures having absolute weights below a certain threshold ξ, for various values of ξ, and present the results in terms of both accuracy and testing time. We begin by running `OSL`α on all 19 sequences of the dataset and present a histogram for each CE representing the distribution of weights learned (Figure 3). The histograms inform us about the portion of the theory that will be pruned for each ξ value. Note that there is a larger number of clauses with weight values in the range $(-1, 1)$. Some of these clauses may be pruned in order to simplify the model without significantly hurting the accuracy, but yielding better inference

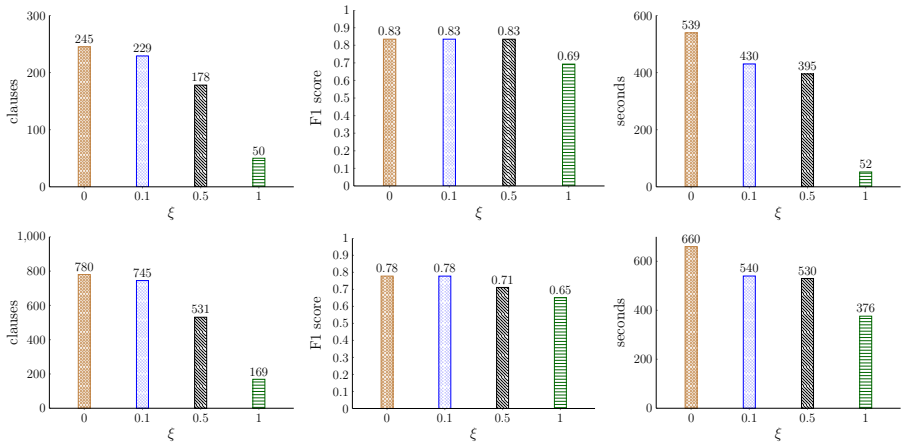

Fig. 4: Effect in the number of clauses learned (left), accuracy (center), and test time (right) as ξ increases for the `meet` CE (top) and the `move` CE (bottom).

times. We pruned the resulting structure for 3 distinct values of ξ and present the results obtained over 10 folds.

Figure 4 presents the reduction in the number of clauses in the resulting theory and the effect in accuracy and testing time as ξ increases. It is worth noting that ξ=0.5 results in a slight reduction in accuracy for `move` and no reduction for `meet`, but test time is improved a lot. Therefore, we can safely prune a subset of the resulting theory in order to improve inference performance.

6 Conclusions and Future Work

We presented the OSLα structure learner for MLNs that exploits background knowledge and uses the MLN–EC axioms to construct CE definitions. The use of MLN–EC axioms allows OSLα to constrain the space of possible structures (i.e., hypergraph) and search only for clauses having characteristics imposed by these axioms. OSLα considers both types of incorrectly predicted CEs (false positives and negatives). Experimental results in activity recognition using a real-world benchmark dataset showed that OSLα outperforms event recognition based on manual rules, and, in some cases, weighted manual definitions. Moreover, OSLα outperforms OSL by learning CE definitions orders of magnitude faster.

We are exploring several directions for future work, such as improving the hypergraph search further using a heuristic or randomized (parallel) graph search procedure, and learning definitions that include negated predicates. We are also studying the problem of structure learning in the presence of unobserved data.

Acknowledgments. This work has been funded by the EU FP7 project SPEEDD (619435).

References

1. Artikis, A., Skarlatidis, A., Paliouras, G.: Behaviour Recognition from Video Content: a Logic Programming Approach. IJAIT, 193–209 (2010)
2. Biba, M., Ferilli, S., Esposito, F.: Discriminative Structure Learning of Markov Logic Networks. In: Proc. of the 18th Int. Conf. on ILP, 59–76 (2008)
3. De Raedt, L.: Logical and Relational Learning, Springer Berlin Heidelberg (2008)
4. De Raedt L., Dehaspe, L.: Clausal discovery. Mach. Learn., 26, 99–146 (1997)
5. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res., 12, 2121–2159 (2011)
6. Domingos, P., Lowd, D.: Markov Logic: An Interface Layer for Artificial Intelligence. Morgan & Claypool Publishers (2009)
7. Getoor, L., Taskar B.: Introduction to Statistical Relational Learning (2007)
8. Heckerman, D.: Learning in Graphical Models. A Tutorial on Learning with Bayesian Networks, 301–354 (1999)
9. Huynh, T.N., Mooney, R.J.: Discriminative structure and parameter learning for markov logic networks. In: Proc of the 25th ICML, 416–423 (2008)
10. Huynh, T.N., Mooney, R.J.: Max-Margin Weight Learning for Markov Logic Networks. In: Proc. of the ECML PKDD, 564–579 (2009)
11. Huynh, T.N., Mooney, R.J.: Online Max-Margin Weight Learning for Markov Logic Networks. In: Proc. of the 11th SDM, 642–651 (2011)
12. Huynh, T.N., Mooney, R.J.: Online Structure Learning for Markov Logic Networks. In: Proc of the ECML PKDD, 81–96 (2011)
13. Khot, T., Natarajan, S., Kersting, K., Shavlik, J.: Gradient-based boosting for statistical relational learning: The markov logic network and missing data cases. Mach. Learn., 100, 75–100 (2015)
14. Kok, S., Domingos P.: Learning the structure of Markov logic networks. In: Proc. of 22nd ICML, 441–448 (2005)
15. Kok, S., Domingos, P.: Learning markov logic network structure via hypergraph lifting. In: Proc. of the 26th ICML, 505–512 (2009)
16. Kok, S., Domingos, P.: Learning markov logic networks using structural motifs. In: Proc. of the 27th Int. Conf. on Machine Learning (ICML), 551–558 (2010)
17. Khosravi, H., Schulte, O., Man, T., Xu, X., Bina, B.: Structure learning for markov logic networks with many descriptive attributes. In: Proc. of the 24th AAAI (2010)
18. McCallum, A.: Efficiently Inducing Features of Conditional Random Fields. In: Proc. of the 19th UAI, 403–410 (2003)
19. Mihalkova, L., Mooney, R.J.: Bottom-up learning of markov logic network structure. In: Proc. of the 24th ICML, 625–632 (2007)
20. Mueller, E.T.: Event Calculus. In: Handbook of Knowledge Representation, vol. 3 of Foundations of Artificial Intelligence, 671–708, Elsevier (2008)
21. Muggleton, S.: Inverse Entailment and Progol. New Gen. Comput., 245–286 (1995)
22. Della Pietra, S., Della Pietra, V., Lafferty, J.: Inducing features of random fields. IEEE Trans. on Pattern Analysis and Machine Intelligence, 380–393, (1997)
23. Quinlan, J.R.: Learning logical definitions from relations. Mach. Learn. (1990)
24. Richardson, M., Domingos, P.: Markov logic networks. Mach. Learn. (2006)
25. Richards, B.L., Mooney, R.J.: Learning relations by pathfinding. In Proc. of the 10th AAAI, 50–55 (1992)
26. Singla, P., Domingos, P.: Discriminative Training of Markov Logic Networks. In: Proc. of the 20th AAAI, 868–873 (2005)
27. Skarlatidis, A., Paliouras, G., Artikis, A., Vouros, G.A.: Probabilistic Event Calculus for Event Recognition. ACM Trans. on Comput. Logic, 16, 1–37 (2015)