Parallel Online Learning of Event Definitions

Nikos Katzouris[1(✉)], Alexander Artikis[1,2], and Georgios Paliouras[1]

[1] National Center for Scientific Research "Demokritos", Athens, Greece
{nkatz,a.artikis,paliourg}@iit.demokritos.gr
[2] Department of Maritime Studies, University of Piraeus, Piraeus, Greece

Abstract. Logic-based event recognition systems infer occurrences of events in time using a set of event definitions in the form of first-order rules. The Event Calculus is a temporal logic that has been used as a basis in event recognition applications, providing among others, direct connections to machine learning, via Inductive Logic Programming (ILP). OLED is a recently proposed ILP system that learns event definitions in the form of Event Calculus theories, in a single pass over a data stream. In this work we present a version of OLED that allows for parallel, online learning. We evaluate our approach on a benchmark activity recognition dataset and show that we can reduce training times, while achieving super-linear speed-ups on some occasions.

1 Introduction

Event recognition systems [9] process sequences of *simple events*, such as sensor data, and recognize *complex events*, i.e. events that satisfy some pattern. Logic-based systems for event recognition [6] typically use a knowledge base of first-order rules to represent complex event patterns and a reasoning engine for pattern matching in the incoming data stream. The Event Calculus (EC) [19] has been used as the basis for event recognition systems [4,23], offering direct connections to machine learning, via Inductive Logic Programming (ILP) [8].

Event recognition applications deal with noisy data streams [1]. Methods that learn from such streams typically build a decision model by a single pass over the input [13]. OLED (Online Learning of Event Definitions) [18] is an ILP system that learns event definitions in the form of EC theories in a single pass over a relational data stream. OLED learns clauses in top-down manner, by gradually specializing an over-general clause using literals from a bottom clause. Its single-pass strategy is based on the Hoeffding bound [15], a statistical tool that allows to build decision models by approximating their quality on the entire input from a small subset of it. We present an extension of OLED, that allows for learning a theory in an online and *parallel* fashion. Our approach is based on a simple parallelization scheme of the core OLED functionality. In the proposed parallelization strategy, a clause is evaluated in parallel on sub-streams of the input stream and its independent scores are combined whenever a specialization

N. Lachiche and C. Vrain (Eds.): ILP 2017, LNAI 10759, pp. 78–93, 2018.
https://doi.org/10.1007/978-3-319-78090-0_6

decision must be made. We present an evaluation of our approach on a benchmark activity recognition dataset and show that it can reduce training times, while it is also capable of super-linear speed-ups on some occasions.

The rest of this paper is structured as follows: In Sect. 2 we present some background on the EC. In Sect. 3 we present OLED and in Sect. 4 we present its parallel version. In Sect. 5 we present our experimental results, while in Sect. 6 we discuss related work. Finally, in Sect. 7 we discuss some directions for future work and conclude.

2 Background

The Event Calculus (EC) [19] is a temporal logic for reasoning about events and their effects. Its ontology consists of *time points* (integer numbers); *fluents*, i.e. properties that have different values in time; and events, i.e. occurrences in time that may alter fluents' values. The axioms of the EC incorporate the *common sense law of inertia*, according to which fluents persist over time, unless they are affected by an event. We use a simplified version of the EC that has been shown to suffice for event recognition [4]. The basic predicates and its domain-independent axioms are presented in Table 1. Axiom (1) states that a fluent F holds at time T if it has been initiated at the previous time point, while Axiom (2) states that F continues to hold unless it is terminated. Definitions for $\mathsf{initiatedAt}/2$ and $\mathsf{terminatedAt}/2$ predicates are given in an application-specific manner by a set of *domain-specific* axioms.

We illustrate our approach using the task of activity recognition, as defined in the CAVIAR project[1]. The CAVIAR dataset consists of videos where actors perform some activities. Manual annotation (performed by the CAVIAR team) provides ground truth for two activity types. The first type corresponds to simple events and consists of knowledge about the activities of a person at a certain video frame/time point, such as *walking*, or *standing still*. The second activity type corresponds to complex events and consists of activities that involve more

Table 1. The basic predicates and domain-independent axioms of EC.

Predicate	Meaning
$\mathsf{happensAt}(E, T)$	Event E occurs at time T
$\mathsf{initiatedAt}(F, T)$	At time T a period of time for which fluent F holds is initiated
$\mathsf{terminatedAt}(F, T)$	At time T a period of time for which fluent F holds is terminated
$\mathsf{holdsAt}(F, T)$	Fluent F holds at time T
Axioms	
$\mathsf{holdsAt}(F, T+1) \leftarrow$ $\quad \mathsf{initiatedAt}(F, T)$	$\mathsf{holdsAt}(F, T+1) \leftarrow$ $\quad \mathsf{holdsAt}(F, T),$ $\quad \mathsf{not\ terminatedAt}(F, T)$

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

than one person, e.g. two people *meeting each other*, or *moving together*. The goal is to recognize complex events as combinations of simple events and additional contextual knowledge, such as a person's direction and position.

Table 2(a) presents some example CAVIAR data, consisting of a narrative of simple events in terms of happensAt/2, expressing people's short-term activities, and context properties in terms of holdsAt/2, denoting people' coordinates and direction. Table 2(a) also shows the annotation of complex events (long-term activities) for each time-point in the narrative. Negated complex events' annotation is obtained via the closed world assumption (although both positive and negated annotation atoms are presented in Table 2, to avoid confusion). Table 2(b) presents two domain-specific axioms in the EC.

Our goal is to learn definitions of complex events in terms of initiation and termination conditions, as in Table 2(b). In the learning setting that we assume the training data consist of Herbrand interpretations, i.e. sets of true ground atoms, as in Table 2(a). Positive examples are annotation atoms contained in such interpretations, while negative examples are false annotation atom instances generated via the closed world assumption. Given a set of training interpretations $\mathcal{I}$, some background theory B, which in our case consists of the domain-independent axioms of the EC, and some language bias M, the goal is to learn a theory H that fits the training data well, i.e. it accounts for as many positive examples and as few negative examples as possible. Formally, given a theory H and an interpretation I, let M_I^H denote a model of $B \cup H \cup I$ and $annotation(I)$ denote the annotation atoms of I. Although different semantics are possible, in this work by "model" we mean a stable model. Also, let $positives(H, I)$

Table 2. (a) Example data from activity recognition. For example, at time point 1 person with id_1 is *walking*, her (x, y) coordinates are $(201, 454)$ and her direction is $270°$. The annotation for the same time point states that persons with id_1 and id_2 are not moving together, in contrast to the annotation for time point 2. **(b)** An example of two domain-specific axioms in the EC. E.g. the first clause dictates that *moving together* between two persons X and Y is initiated at time T if both X and Y are walking at time T, their euclidean distance is less than 25 pixel positions and their difference in direction is less than 45°. The second clause dictates that *moving together* between X and Y is terminated at time T if one of them is standing still at time T (exhibits an inactive behavior) and their euclidean distance at T is greater that 30.

(a)

Narrative for time 1:	**Narrative for time 2:**
happensAt$(walk(id_1), 1)$.	happensAt$(walk(id_1), 2)$.
happensAt$(walk(id_2), 1)$.	happensAt$(walk(id_2), 2)$.
holdsAt$(coords(id_1, 201, 454), 1)$.	holdsAt$(coords(id_1, 201, 454), 2)$.
holdsAt$(coords(id_2, 230, 440), 1)$.	holdsAt$(coords(id_2, 227, 440), 2)$.
holdsAt$(direction(id_1, 270), 1)$.	holdsAt$(direction(id_1, 275), 2)$.
holdsAt$(direction(id_2, 270), 1)$.	holdsAt$(direction(id_2, 278), 2)$.
Annotation for time 1:	**Annotation for time 2:**
not holdsAt$(move(id_1, id_2), 1)$	holdsAt$(move(id_1, id_2), 2)$

(b)

Two Domain-specific axioms:

initiatedAt$(move(X, Y), T) \leftarrow$
happensAt$(walk(X), T)$,
happensAt$(walk(Y), T)$,
$distLessThan(X, Y, 25, T)$,
$dirLessThan(X, Y, 45, T)$.

terminatedAt$(move(X, Y), T) \leftarrow$
happensAt$(inactive(X), T)$,
$distMoreThan(X, Y, 30, T)$.

(resp. $negatives(H, I)$) be the set of complex event instances a with the property $\alpha \in M_I^H \cap annotation(I)$ (resp. $\alpha \in M_I^H \setminus annotation(I)$). The goal then is to learn a theory H with the property

$$\underset{H \in \mathcal{L}(M)}{\text{argmax}}(\sum_{I \in \mathcal{I}} |positives(H, I)| - |negatives(H, I)|)$$

where $\mathcal{L}(M)$ denotes the hypothesis language defined by the language bias M. The language bias that we assume is mode declarations [20].

3 The OLED System

OLED [18] learns a theory by joining together independently-constructed clauses, each of which is learnt in an online fashion. It relies on the Hoeffding bound [15] to approximate the quality of a clause on the entire input using only a subset of the data. Given a random variable X with range in $[0, 1]$ and an observed mean $\overline{X}$ of its values after n independent observations, the Hoeffding Bound states that, with probability $1 - \delta$, the true mean $\hat{X}$ of the variable lies in an interval $(\overline{X} - \epsilon, \overline{X} + \epsilon)$, where $\epsilon = \sqrt{\frac{ln(1/\delta)}{2n}}$. In other words, the true average can be approximated by the observed one with probability $1 - \delta$, given an error margin ϵ that becomes smaller as the number of observations n increases.

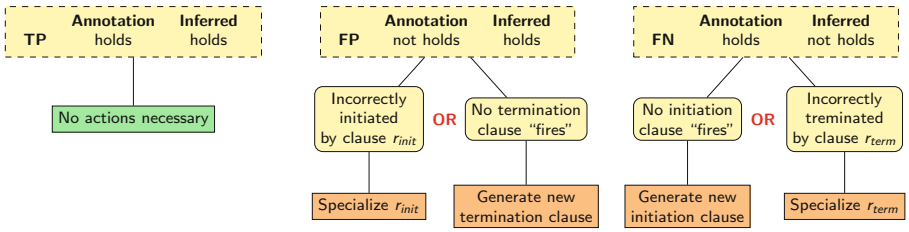
OLED learns a clause in a top-down fashion, by specializing it using literals from a bottom clause [8]. The Hoeffding bound is utilized in the specialization process as follows. Given a clause evaluation function G and some clause r, OLED evaluates r and all of its candidate specializations on training interpretations that stream-in, counting positive and negative examples in these interpretations that are covered by each of these clauses. Assume that after n examples, r_1 is r's specialization with the highest observed mean G-score $\overline{G}$ and r_2 is the second-best one, i.e. $\Delta\overline{G} = \overline{G}(r_1) - \overline{G}(r_2) > 0$. Then by the Hoeffding bound we have that for the true mean of the scores' difference $\Delta\hat{G}$ it holds that $\Delta\hat{G} > \Delta\overline{G} - \epsilon$, with probability $1 - \delta$, where $\epsilon = \sqrt{\frac{ln(1/\delta)}{2n}}$. Hence, if $\Delta\overline{G} > \epsilon$ then $\Delta\hat{G} > 0$, implying that r_1 is indeed the best specialization, with probability $1 - \delta$. In order to decide which specialization to select, it thus suffices to accumulate example counts from the incoming interpretations until $\Delta\overline{G} > \epsilon$. These interpretations need not be stored or reprocessed. Each interpretation is processed once to extract the necessary statistics for calculating G-scores and is subsequently discarded, thus giving rise to an online (single-pass) clause construction strategy. To ensure that no clause r is replaced by a specialization of lower quality, r itself is also considered as a potential candidate along with its specializations, ensuring that specializing r is a better decision, with probability $1 - \delta$, than not specializing it at all.

The default specialization process follows a FOIL-like, hill-climbing strategy, where a single literal is added to a clause at each specialization step. However, OLED supports different specialization strategies as well, e.g. by allowing to simultaneously try all specializations up to a given clause length, or by supporting user-defined, TILDE-like look-ahead specifications [7].

To calculate G-scores, each clause r is equipped with a true positive (TP), a false positive (FP) and a false negative (FN) counter, whose values are updated accordingly as r gets evaluated on training interpretations that stream-in. True negative counts are not taken into account, since the annotation for complex events is acquired via the closed world assumption. Although different scoring functions may be plugged into OLED, in this work we use precision to score initiation clauses and recall to score termination clauses, as in [18]. Moreover, OLED supports a clause pruning mechanism, that allows to remove low-quality clauses (e.g. clauses that have been generated from noisy interpretations) and a tie-breaking mechanism, that allows to randomly select between equally good specializations. We refer to [18] for more details on these features.

In the general case, a theory learnt by OLED is a collection of clauses constructed with the online mechanism described above. A clause is generated from a positive example in an incoming interpretation, by constructing a bottom clause

(A)



(B)

	TP instance occurs	*FP* instance occurs	*FN* instance occurs
InitiationLearner	Reward all clauses that correctly initiate the complex event.	Penalize all clauses that incorrectly initiate the complex event.	Generate new initiation clause.
TerminationLearner	Reward all clauses that correctly allow the complex event to persist.	Generate new termination clause.	Penalize all clauses that incorrectly terminate the complex event.

Fig. 1. **(A)** Different behaviors of initiation and termination clauses w.r.t. to occurrences of TP, FP and FN complex event instances. Dash-lined boxes explain what it means to encounter a TP, FP, FN complex event instance, in terms of (dis)agreement between the actual label of the instance and the one inferred by the theory. Round-cornered boxes describe the causes of FP, FN occurrences w.r.t. the different types of clause (initiation or termination). Regular boxes at the "leaves" of the tree-like structures indicate proper courses of action in order to eliminate FP/FN instances. **(B)** Actions taken by the two different processes that learn initiation and termination clauses in parallel, w.r.t. TP, FP, FN complex event occurrences. These actions are in accordance with the indicated actions in **(A)** (leaves of the trees). "Rewarding" a clause refers to increasing the TP count of the clause, while "penalizing" a clause refers to increasing its FP or FN counts. Penalizing clauses reduces their score, it therefore contributes to their specialization after a sufficient number of examples has been seen.

$\perp$ from that instance and adding the empty-bodied clause "$r = head(\perp) \leftarrow$" to theory H. From that point on, r is gradually specialized by the addition of literals from $\perp$ to its body. New clauses are added to H whenever existing clauses in H become too specific to account for positive examples in new incoming interpretations. Bottom clause construction is preceded by an abductive process that handles the fact that target predicates (initiatedAt/2 and terminatedAt/2) differ from observed annotation predicates (holdsAt/2). We refer to [18] for more details.

When learning domain-specific axioms in the Event Calculus, the aforementioned generic theory construction strategy must be modified to account for the fact that initiation and termination clauses behave differently w.r.t. encountered TP, FP and FN complex event instances. A description of this behavior is illustrated in Fig. 1(A). To handle this behavior, initiation and termination clauses are learnt separately, by two parallel processes, each of which runs the core OLED Algorithm. The input stream is forwarded to each of these processes. Figure 1(B) presents the different actions that each learner takes whenever it encounters TP, FP and FN instances.

4 A Parallel Version of OLED

We now proceed to the description of a data parallel version of OLED, which we henceforth denote by p-OLED. The parallelization strategy is based on evaluating a clause and its candidate specializations on incoming interpretations by distributing the workload across multiple processing nodes that operate on sub-streams of the input stream. When a node is about to specialize or remove a clause r, it consults its peer nodes and combines their evaluation results for r with its own, so that a more informed decision is made. We next describe this strategy in more detail.

4.1 Main Operations of the Parallel OLED Strategy

We assume that learning is performed by a set $\mathcal{N}$ of independent processing nodes. Each node handles a sub-stream S_i of training interpretations, generated from an input stream $\mathcal{S}$, according to some data distribution scheme. For instance, the data from $\mathcal{S}$ may be distributed to the processing nodes in $\mathcal{N}$ in a "round-robin" manner, or by using specific data attributes as "key" in the distribution process. Processing nodes communicate by exchanging messages and they collaborate in order to learn a theory H in parallel. In particular, p-OLED differs from the sequential algorithm in the following respects:

New clause generation: When a node N_i generates a new clause r, it broadcasts r to all other nodes in $\mathcal{N}$, via an `AddNewClause`(r) message (see Table 3 for the main types of message of parallel OLED). Each node that receives such a message adds clause r to its own theory and starts scoring r, and its candidate

Table 3. The main messages exchanged between data processing nodes in parallel OLED.

Message	Conditions for message broadcast	Actions upon message receipt
`AddNewClause`(r)	Generation of clause r	Add r to local theory
`SpecializeRequest`(r_{id})	Clause with id r_{id} is about to be specialized (the Hoeffding test for this clause has succeeded)	Reply to the sender by the local TP, FP, FN, E counts for clause with id r_{id} and for each of its candidate specializations
`SpecializeReply`($args$), where $args = \langle r_{id}, TP, FP, FN, E\rangle$	Reply to a specialization request message for the clause with id r_{id}	Add the received counts for the corresponding clause to the local ones and repeat the Hoeffding test
`Replace`(r_{id}, r')	Clause with id r_{id} has been specialized to clause r'	Replace clause with id r_{id} by r' in local theory
`PruneRequest`(r_{id})	Clause with id r_{id} is about to be pruned	Reply to the sender by the local TP, FP, FN counts for clause with id r_{id}, as well as the period for which r remains (locally) unchanged
`PruneReply`($args$), where $args = \langle r_{id}, TP, FP, FN, T\rangle$, T being the period for which the clause with id r_{id} remained unchanged at the sender node	Reply to a prune request message	Add the received counts for the corresponding clause to the local ones and repeat the clause removal test
`Remove`(r_{id})	Clause with id r_{id} has been pruned	Remove clause with id r_{id} from local theory

specializations on its local data stream. As in the sequential version of OLED, a new clause r consists of an initially empty-bodied clause "$head(\perp) \leftarrow$", where $\perp$ is a bottom clause generated at N_i, which is subsequently used to gradually specialize r.

Clause specialization: When a node N_i is about to specialize a clause r, i.e. when OLED's Hoeffding test for clause r succeeds, locally at N_i, node N_i sends a `SpecializeRequest`(r_{id}) message to all other nodes, where r_{id} is a unique identifier of clause r, common to all copies of r shared among processing nodes. Upon receiving such a message, each node uses r_{id} to retrieve its own evaluation statistics for clause r and its candidate specializations, which are sent over to the requesting node N_i. These statistics consist of TP, FP, FN and E counts for clause r and its candidate specializations, where by E we denote the number of examples on which a clause has been evaluated so far (number of groundings of target predicates). The received counts for clause r and its specializations are combined with node N_i's local counts as follows (we describe the process for clause r only, but it is similar for each of its specializations). Denoting by TP_r^j, FP_r^j, FN_r^j and E_r^j the respective counts for clause r, received from node $N_j \in \mathcal{N}, j \neq i$, the current node N_i updates r's counts accordingly, by increasing

r's local counts with those received from other nodes. For instance, the new TP count for clause r in node N_i becomes $TP_r^i = TP_r^i + \sum_{N_j \in \mathcal{N}} TP_r^j$.

Each processing node $N_i \in \mathcal{N}$ maintains a record, for each clause r in its theory and each one of r's specializations, that contains the exact counts previously received for them, from each node $N_j \in \mathcal{N}, j \neq i$. When node N_i receives a set of new TP_r^j, FP_r^j, FN_r^j and E_r^j counts for clause r from node, the respective previous counts are subtracted from the new ones, to avoid over-scoring r with counts that have already been taken into account in previous updates. The same holds for r's specializations.

Once individual clause evaluation statistics are combined as described above, node N_i repeats the Hoeffding test for clause r to assess if the test still succeeds after the accumulated counts have been taken into account. If it does, clause r is replaced in H, the current theory at node N_i, by its best-scoring specialization r' that results from the Hoeffding test. Then, node N_i sends out a $\texttt{Replace}(r_{id}, r')$ message to all other nodes, instructing them to also replace their local copy that corresponds to r_{id} in their own theories with r'. If, on the other hand, the Hoeffding test fails at node N_i after the updated counts are taken into account, clause r is not specialized.

Clause pruning: For a clause r to be removed, two conditions must hold: First, clause r must be unchanged (not specialized) for a sufficiently long period, which, in the single-core version of OLED, is set to the average number of examples for which the Hoeffding test succeeds, i.e. the average value of $n = \mathcal{O}(\frac{1}{\epsilon^2} ln \frac{1}{\delta})$ that has resulted in clause specializations so far. Second, from that point on where clause r remains unchanged, a sufficiently large number of data must be seen, in order to use a Hoeffding test to infer that, with probability $1 - \delta$, the quality of clause r is below the pruning threshold, i.e. a user-defined lower bound on the quality of acceptable clauses.

In p-OLED, each node uses the above heuristics to decide locally whether a clause r should be pruned. Once it has seen enough data from its own stream to make that decision for clause r, it sends a $\texttt{PruneRequest}(r_{id})$ message to all other nodes. Each node that receives such a message sends back to the requesting node the necessary information (period for which clause r remains unchanged and TP, FP, FN and E_r counts for clause r), which node N_i uses to re-assess whether clause r should be pruned, based on the global view of clause r, obtained by combining r's separate evaluations from all processing nodes. If node N_i eventually decides to prune clause r, it sends a $\texttt{Remove}(r_{id})$ to all other nodes, which instructs them to also remove clause r from their theories.

4.2 Decentralized Coordination

Each processing node in p-OLED operates autonomously and there is no centralized coordination. This may result in undesired behavior, therefore some extra actions are in order, at an implementation level, to avoid such behavior. We next discuss such issues and outline the way that p-OLED handles them.

Clause specialization requires some coordination between processing nodes. For instance, assume that node N_j handles a `SpecializeRequest` for some clause r, sent from a node N_i. N_j sends r's evaluation statistics to the requesting node N_i and it subsequently continues to process data from its local training stream. This implies that during the time taken for node N_i to decide on r's specialization (receive the statistics for r and its candidate specializations from all nodes and repeat the Hoeffding test), node N_j continues to evaluate clause r on its own data. It is possible that during this time the Hoeffding test for clause r succeeds at node N_j, in which case it will attempt to specialize r. This is unnecessary, since r's specialization is already under assessment at node N_i. To avoid this behaviour and ensure that a potential specialization of a clause r is handled by a single node at a time, each recipient node of a `SpecializeRequest` message "marks" the clause in question as a specialization candidate. For a marked clause r, all potential specialization attempts are temporarily suspended, until a "verdict" for this clause is received from the node that is currently attempting to specialize clause r.

The above strategy is insufficient in cases where the Hoeffding test for clause r succeeds at more than one processing nodes simultaneously, or at a very close temporal proximity. In such cases, a node N_i may need to handle a `SpecializeRequest` for some clause r, while currently attempting itself to specialize r, implying that two nodes are attempting to specialize the same clause simultaneously. Consider for instance a situation where node N_i has just finished processing an interpretation where the Hoeffding test for clause r succeeded, while in the meantime, a `SpecializeRequest` message for the same clause r, sent from some other node N_j, has been enqueued in N_i's message queue. To resolve conflicts in cases like these, a priority order is imposed beforehand on all processing nodes, using each node's index k, $1 \leq k \leq |\mathcal{N}|$. Nodes of higher index are prioritized to specialize a clause r over nodes of lower index. That is, a node of index k abandons its effort to specialize a clause r whenever it encounters a `SpecializeRequest` message for the same clause, received from a node with index $k' > k$. Similarly, nodes of higher index do not serve specialization requests for a clause r, received from nodes of lower index, in case they themselves are already attempting to specialize r.

A similar coordination mechanism is used to ensure that a potential removal of a low-quality clause during pruning is handled by a single node at a time.

Another cause for unwanted behaviour is related to delays in message passing. For instance, a node N_i may be currently carrying out an intensive, time-consuming task (e.g. processing a large and complex interpretation), while some other node N_j is expecting N_i's reply on a `SpecializeRequest` message, in order to specialize some clause r. This results in N_j "wasting data", since it keeps processing new interpretations during this time. These data could have been used to evaluate new specializations for clause r, had node N_i responded in a timely fashion. To avoid situations like these, in practice each node uses a time-out parameter t to handle the replies of its peers nodes, considering only the replies received within the time-out. The time-out parameter is adapted during the learning process according to the mean processing time per training interpretation.

5 Empirical Evaluation

We present an experimental evaluation of our approach on CAVIAR (described in Sect. 2), a benchmark dataset for activity recognition. CAVIAR contains 282,067 training interpretations with a mean size of 25 atoms each. p-OLED is implemented in the Scala programming language. It uses Clingo[2] as its main reasoning component and Scala's akka Actors library[3] to model the behavior of a processing node and implement message passing. The code and data of the empirical analysis are available online[4]. All experiments were conducted on a Debian Linux machine with a 3.6 GHz processor (4 cores and 8 threads) and 16 GB of RAM.

The purpose of our first experiment was to compare p-OLED with its monolithic counterpart. We performed learning with 1, 2, 4 and 8 processing threads (each representing a processing node) for constructing the definitions of two complex events, related to two persons *meeting each other* or *moving together*. We used tenfold cross-validation with an 80%–20% training-testing ratio. CAVIAR contains 6,272 positive interpretations for *moving* (i.e. interpretations where *moving* occurs) and 3,722 positive interpretations for *meeting*, forming respectively 12 positive sequences for *moving* and 11 positive sequences for *meeting* (a positive (resp. negative) sequence encompasses a time interval where a complex event holds (resp. does not hold) continuously). The testing set for each fold of the cross-validation process consisted of 2 positive sequences per complex event, plus negative sequences amounting to the 20% of the total negatives in the dataset, while the remaining positive and negative sequences where used for training. For this experiment positive and negative sequences in the training set were evenly distributed across the different processing nodes, so that all nodes were fed with approximately the same number of positive and negative examples. In each fold of the cross-validation process, the training interpretations were presented to each processing node in a random order. The parameters for both the sequential and the parallel version of OLED was $\delta = 10^{-5}$ and clause pruning threshold set to 0.65 for *moving* and 0.8 for *meeting*. These values were chosen empirically based on previous experiments with OLED on the CAVIAR dataset [18]. The pruning threshold values refer to precision for initiation clauses and recall for termination clauses, which were used as scoring functions in this experiment.

We also created a larger version of CAVIAR, in order to evaluate our algorithms on a more demanding learning task. This dataset consists of 10 copies of the original CAVIAR dataset, where each copy differs from the others only in the constants referring to the tracked entities (persons, objects) that appear in simple and complex events. This dataset contains 100 different tracked entities, as compared to only 10 entities of the original CAVIAR dataset. In each copy of the dataset, the coordinates of each entity p differ by a fixed offset from the coordinates of the entity of the original dataset that p mirrors. The setting for the x10-CAVIAR experiment was as described above.

[2] http://potassco.sourceforge.net/.
[3] http://akka.io/.
[4] https://github.com/nkatzz/OLED.

The results from our experiment with CAVIAR and x10-CAVIAR are presented in Table 4(A) and (B) respectively, in the form of averages (over the ten runs of the cross-validation process) for training time, F_1-score and theory size (total number of literals), as well as average number of exchanged messages. F_1-scores were obtained by micro-averaging results from each fold. We also present F_1-scores and theory sizes for hand-crafted theories for the two target complex events. The hand-crafted theories may be considered as the standard in this domain and they are presented in [5]. They are available online[5], in addition to learnt theories for *meeting* and *moving* with OLED. Table 4 also presents the achieved speed-ups for p-OLED, defined as T_1/T_n, where T_1 and T_n are respectively the training times of a monolithic and a parallel learner that uses n cores. The speed-up is linear if it's approximately equal to n, for each n, while it is sub-linear (resp. super-linear) if it is smaller (resp. greater) than n.

Starting with the results from the CAVIAR dataset (Table 4(A)), we see that p-OLED constructed theories of slightly higher F_1-score for *meeting*, as compared to its single-core counterpart. In the monolithic setting, OLED postpones

Table 4. (A) Experimental results from the CAVIAR dataset; **(B)** Experimental results from the x10-CAVIAR dataset.

		#cores	Time (sec)	Speed-up	F_1-score	Theory size	#Msgs
(A)	***Meet***	1	46	–	0.798	28	–
		2	18	2.5	**0.818**	31	**75**
		4	15	**3**	0.805	34	168
		8	**15**	**3**	0.802	35	358
	HandCrafted	–	–	–	0.700	**24**	–
	Move	1	68	–	**0.744**	**21**	–
		2	31	2.1	0.740	**21**	**58**
		4	27	2.5	0.739	**21**	112
		8	**26**	**2.6**	0.743	23	228
	HandCrafted	–	–	–	0.732	28	–
(B)	***Meet***	1	7588	–	**0.834**	36	–
		2	2144	3.5	**0.834**	36	**78**
		4	1682	4.5	**0.834**	36	158
		8	**912**	**8.3**	0.832	36	342
	HandCrafted	–	–	–	0.700	**24**	–
	Move	1	7898	–	**0.758**	34	–
		2	2312	3.4	0.753	34	**82**
		4	1788	4.4	0.756	34	164
		8	**966**	**8.1**	0.753	34	322
	HandCrafted	–	–	–	0.732	**28**	–

[5] http://users.iit.demokritos.gr/~nkatz/CAVIAR-theories/.

the generation of new clauses, up to the point where existing clauses become too specific to account for new examples in the incoming interpretations. During this time, interpretations which may result in a good clause (recall that OLED learns by "encoding" interpretations into bottom clauses), are "skipped", i.e. they are not used for learning new clauses, since they are covered by existing ones. In contrast, the data distribution in p-OLED resulted in cases where interesting interpretations that would have been missed in the monolithic setting, are actually used for learning. A similar effect was not observed for *moving*, which has a simpler definition than *meeting*.

Regarding training times, OLED achieves a significant speed-up for both complex events, by moving from sequential learning to learning with 2 cores, but from that point on, training times do not improve proportionally to the number of cores, resulting in sub-linear speed-ups. This is not the case however in the x10-CAVIAR experiment (Table 4(B)), where p-OLED achieves super-linear speed-ups. The x10-CAVIAR dataset consists of larger training interpretations, each containing an increased number of domain constants and ground literals. Such interpretations are significantly harder to be reasoned upon, as indicated by the exponential growth in training times in the x10-CAVIAR experiment. Therefore, the contrast between the speed-up patterns of the regular CAVIAR and the significantly larger x10-CAVIAR version seems to be in line with the fact that gains by parallelizing an ILP algorithm are often observed only when significant data volumes are involved [12,28,30]. Additionally, the reported behavior seems to imply that the gain in efficiency of our proposed parallel learning strategy increases with the difficulty of the learning task at hand, in terms of the "unit cost" of processing individual interpretations.

Due to the increase in training data size in the x10-CAVIAR experiment, F_1-scores for all runs (number of cores) are improved as compared to the regular CAVIAR experiment and they seem to converge. For example, in the regular CAVIAR experiment, good rules were often constructed "too-late", from interpretations that were encountered shortly before the data were exhausted. Such rules may be discarded, since OLED (and its parallel version) use a "warm-up" period parameter that controls a minimum number of interpretations a rule must be evaluated on, in order to be included in an output hypothesis. In contrast, in the x10-CAVIAR experiment such problems were avoided, thanks to the increase in training data size.

We performed an additional experiment where the goal was to assess the effect of uneven data distribution on the amount of communication, total training time and F_1-score. The experimental setting was similar to the one described previously, i.e. a tenfold cross-validation process with an $80\% - 20\%$ training-testing ratio. One of the nodes, however, handled a larger data load than its peers. To introduce the imbalance we used an external data distribution processes that takes as input an imbalance parameter k. This process reads the data from disk, in the actual order in which they appear in the CAVIAR videos, and forwards training interpretations to processing nodes as follows: The first k interpretations are forwarded to the first node. Subsequently, each one of the

Table 5. Effects on the imbalance in data load on CAVIAR, using 8 processing nodes.

	Imbalance	Time (sec)	#Msgs	F_1-score	Theory size
Meet	10	**16**	348	0.802	**34**
	50	24	327	**0.808**	**34**
	100	41	**298**	0.799	**34**
Move	10	**24**	231	0.739	22
	50	38	212	0.741	**23**
	100	52	**191**	**0.742**	**23**

following $N-1$ interpretations (N being the number of used nodes) is forwarded to one of the remaining $N-1$ nodes. The next k interpretations are forwarded again to the first node and so on, so that the first node eventually handles k-times more data than its peer nodes. In the process of data distribution, data sequences that are intended to be used for testing are "skipped" (they are not forwarded to any node).

We performed experiments in this setting with 8 processing nodes and three different values for the imbalance parameter $k = 10, 50, 100$. The results are presented in Table 5 in the form of averages from the tenfold cross-validation process for total training time, number of messages, F_1-score (micro-averaged over all folds) and theory size. As the imbalance parameter k grows, training time increases slightly, while the amount of communication drops. The increase in training time may be explained by the "bottleneck" of a single node handling larger data loads sequentially, as the imbalance increases, while the drop in the total number of exchanged messages is due to the fact the majority of the processing nodes, which handle fewer training data, also broadcast fewer messages, as compared to the scenario where data are evenly distributed between nodes. Regarding the F_1-score and the theory size, only small changes are reported with respect to the results of Table 4(A). These differences are attributed to the different order in which training interpretations are presented to p-OLED in the two experiments.

6 Related Work

An overview of existing approaches to learning theories in the Event Calculus with ILP may be found in [16,17] and a discussion on how OLED compares to such approaches may be found in [16,18]. In this section we mainly discuss parallel ILP algorithms, for which a substantial amount of work exists in the literature. A thorough review may be found in [12,30]. Parallel ILP algorithms exploit parallelism across three main axes [12]: Searching through the hypothesis space in parallel (search parallelism); splitting the training data and learning from data subsets (data parallelism); and evaluating candidate clauses in parallel (evaluation/coverage parallelism).

In [28] the authors present a data-parallel version of a standard set-cover loop: Each processing node learns a fragment of the concept definition from a partition of the data, and then these fragments are exchanged between all nodes. Good-enough clauses are kept by all nodes. A cover removal step is subsequently implemented by each core and the set-cover loop continues. Overall, the approach in [28] learns much faster than a sequential algorithm, achieving super-linear speed-ups. A similar approach is proposed in [11], where the training interpretations are split across multiple nodes and searched in parallel, while the best rules from each node are "pipe-lined" to all other nodes.

In [30] the authors use a MapReduce-based framework to parallelize the operation of a classical set-cover ILP algorithm towards both evaluation-parallelism and search-parallelism. In the former case, coverage tests of candidate clauses are performed in parallel, on disjoint partitions of the data. In the latter case, bottom clauses (which are generalized to acquire a hypothesis clause) are generated and searched in a concurrent fashion from more than one "seed" examples. The reducer then selects the best hypothesis clause that results from this process. A similar approach for parallel exploration of independent hypotheses has been proposed in [22], while similar approaches towards parallel coverage tests have been proposed in [10,14]. In [21], the approach of [30] was extended to a framework that is capable of self-regulating the workload of distributing learning costs across multiple nodes. In [25,26] the authors propose a strategy for collaborative learning of action models. The problem is modelled in a multi-agent systems context, where autonomous agents communicate with each other and revise their local models in an effort to establish global consistency of the latter. An important difference of this work is that the communication is based on the exchange of examples, as opposed to clauses, which is the case with p-OLED and most of the works described above. Finally, some work exists in the literature on parallelizing (unsupervised) relational data-mining tasks, such as frequent pattern mining [2,3].

A main difference of the work presented here from the aforementioned approaches to parallel ILP is that they mostly rely on iterative ILP algorithms, which require several passes over the data to compute a hypothesis. In contrast, OLED is an online, single-pass algorithm. In relation to the latter, some work on streaming ILP exists. However, existing approaches are either oriented towards unsupervised tasks like frequent pattern discovery [27], or they rely on propositionalization techniques and off-the-self, online propositional learners [29].

7 Conclusions and Future Work

We presented a parallel version of a recently proposed algorithm for online learning of event definitions in the form of Event Calculus theories. We also presented an experimental evaluation of our approach on a benchmark dataset for activity recognition, which demonstrates that it can reduce training times, while also achieving super-linear speed-ups on some occasions. As future work, we aim to evaluate our approach in a distributed setting and on larger datasets, in terms of

in-situ, geographically distributed learning, as required in maritime monitoring [24]. We also plan to formally analyze the behavior of the proposed approach in terms of communication cost, comparison to its monolithic counterpart in terms of convergence and convergence speed, as well as comparison to similar learning strategies that adopt different communication protocols.

Acknowledgments. This work is funded by the H2020 project datAcron (687591).

References

1. Alevizos, E., Skarlatidis, A., Artikis, A., Paliouras, G.: Probabilistic complex event recognition: a survey. ACM Comput. Surv. (2018, to appear)
2. Appice, A., Ceci, M., Turi, A., Malerba, D.: Sampling very large databases for parallel and distributed relational frequent pattern discovery. In: First International Workshop on Ubiquitous Knowledge Discovery Workshop (2008)
3. Appice, A., Ceci, M., Turi, A., Malerba, D.: A parallel, distributed algorithm for relational frequent pattern discovery from very large data sets. Intell. Data Anal. **15**(1), 69–88 (2011)
4. Artikis, A., Sergot, M., Paliouras, G.: An event calculus for event recognition. IEEE Trans. Knowl. Data Eng. **27**(4), 895–908 (2015)
5. Artikis, A., Skarlatidis, A., Paliouras, G.: Behaviour recognition from video content: a logic programming approach. Int. J. Artif. Intell. Tools **19**(2), 193–209 (2010)
6. Artikis, A., Skarlatidis, A., Portet, F., Paliouras, G.: Logic-based event recognition. Knowl. Eng. Rev. **27**(4), 469–506 (2012)
7. Blockeel, H., De Raedt, L.: Top-down induction of first-order logical decision trees. Artif. Intell. **101**(1), 285–297 (1998)
8. De Raedt, L.: Logical and Relational Learning. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68856-3
9. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications Co., Greenwich (2010)
10. Fidjeland, A.K., Luk, W., Muggleton, S.H.: Customisable multi-processor acceleration of inductive logic programming. In: Latest Advances in Inductive Logic Programming, pp. 123–141 (2014)
11. Fonseca, N.A., Silva, F.M.A., Costa, V.S., Camacho, R.: A pipelined data-parallel algorithm for ILP. In: 2005 IEEE International Conference on Cluster Computing (CLUSTER 2005), Boston, Massachusetts, USA, 26–30 September 2005, pp. 1–10 (2005)
12. Fonseca, N.A., Srinivasan, A., Silva, F., Camacho, R.: Parallel ILP for distributed-memory architectures. Mach. Learn. **74**(3), 257–279 (2009)
13. Gama, J.: Knowledge Discovery from Data Streams. CRC Press, Florida (2010)
14. Graham, J.H., David Page Jr., C., Kamal, A.H.: Accelerating the drug design process through parallel inductive logic programming data mining. In: 2nd IEEE Computer Society Bioinformatics Conference, CSB 2003, Stanford, CA, USA, 11–14 August 2003, pp. 400–402 (2003)
15. Hoeffding, W.: Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. **58**(301), 13–30 (1963)
16. Katzouris, N.: Scalable relational learning for event recognition. Ph.D. thesis, University of Athens (2017). http://users.iit.demokritos.gr/~nkatz/papers/nkatz-phd.pdf

17. Katzouris, N., Artikis, A., Paliouras, G.: Incremental learning of event definitions with inductive logic programming. Mach. Learn. **100**(2–3), 555–585 (2015)
18. Katzouris, N., Artikis, A., Paliouras, G.: Online learning of event definitions. TPLP **16**(5–6), 817–833 (2016)
19. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Gener. Comput. **4**(1), 67–95 (1986)
20. Muggleton, S.: Inverse entailment and progol. New Gener. Comput. **13**(3&4), 245–286 (1995)
21. Nishiyama, H., Ohwada, H.: Yet another parallel hypothesis search for inverse entailment. In: ILP (2015)
22. Ohwada, H., Mizoguchi, F.: Parallel execution for speeding up inductive logic programming systems. In: Arikawa, S., Furukawa, K. (eds.) DS 1999. LNCS (LNAI), vol. 1721, pp. 277–286. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-46846-3_25
23. Paschke, A., Bichler, M.: Knowledge representation concepts for automated SLA management. Decis. Support Syst. **46**(1), 187–205 (2008)
24. Patroumpas, K., Alevizos, E., Artikis, A., Vodas, M., Pelekis, N., Theodoridis, Y.: Online event recognition from moving vessel trajectories. GeoInformatica **21**(2), 389–427 (2017)
25. Rodrigues, C., Soldano, H., Bourgne, G., Rouveirol, C.: A consistency based approach of action model learning in a community of agents. In: International conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2014, Paris, France, 5–9 May 2014, pp. 1557–1558 (2014)
26. Rodrigues, C., Soldano, H., Bourgne, G., Rouveirol, C.: Multi agent learning of relational action models. In: ECAI 2014 - 21st European Conference on Artificial Intelligence, 18–22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014), pp. 1087–1088 (2014)
27. Silva, A., Antunes, C.: Multi-relational pattern mining over data streams. Data Min. Knowl. Disc. **29**(6), 1783–1814 (2015)
28. Skillicorn, D.B., Wang, Y.: Parallel and sequential algorithms for data mining using inductive logic. Knowl. Inf. Syst. **3**(4), 405–421 (2001)
29. Srinivasan, A., Bain, M.: Relational models with streaming ILP. In: ILP (2013)
30. Srinivasan, A., Faruquie, T.A., Joshi, S.: Data and task parallelism in ILP using mapreduce. Mach. Learn. **86**(1), 141–168 (2012)