Parallel Online Event Calculus Learning for Complex Event Recognition

Nikos Katzouris[a], Alexander Artikis[b,a], Georgios Paliouras[a]

[a]*National Center for Scientific Research "Demokritos", Athens, Greece*
[b]*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. We present two strategies for parallel online learning with OLED. We evaluate our proposed approaches on three datasets from the domains of activity recognition and maritime surveillance and show that they can significantly reduce training times, while they are capable of achieving super-linear speed-ups under certain circumstances.

Keywords: Online Learning; Parallel Learning; Event Calculus; Event Recognition;

1. Introduction

Event recognition systems [1, 2, 3] process sequences of *simple events*, such as sensor data, and recognize *complex events*, i.e. events that satisfy some pattern. Logic-based event recognition systems typically use a knowledge base of first-order rules to represent complex event patterns and a reasoning engine to detect such patterns in a data stream [4]. The Event Calculus (EC) [5] has been used as the basis for event recognition systems [6], offering direct connections to machine learning, via Inductive Logic Programming (ILP) [7].

Event recognition applications deal with noisy data streams. Methods that learn from such streams typically build a decision model by a single pass over the input [8]. OLED (Online Learning of Event Definitions) [9] is an ILP system that learns event definitions in the form of EC theories in a single pass over a relational data stream. In an effort to pave the way for relational learning in high-velocity data streams, we present two extensions of OLED, that allow for learning in an online and parallel

*Corresponding author
Email address: support@elsevier.com (Georgios Paliouras)
URL: www.elsevier.com (Nikos Katzouris)

fashion, from disjoint data streams. Both strategies assume a set of processing nodes that communicate via message passing. Our first strategy is a "synchronous" one, in the sense that all processing nodes collaborate to learn a rule simultaneously. The second strategy is an "asynchronous" one, where each node learns independently and the best rules from each node are shared between all nodes, where the rules are further evaluated and revised if necessary. In comparison, the latter strategy has a lower communication overhead, but also a slower rate of converging to a high-quality theory. We present a comparative evaluation of our approaches on three datasets from the domains of activity recognition and maritime surveillance and we show that they can significantly reduce training times, while they are capable of achieving super-linear speed-ups on some occasions.

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

2. Background and Running Example

We assume a logic programming setting as in [10], where not in front of logical literals denotes Negation as Failure. Rules are denoted by $\alpha \leftarrow \delta_1, \ldots, \delta_n$, where α is a non-negated literal, called the *head* (consequent) of the rule and $\{\delta_1, \ldots, \delta_n\}$ are (possibly negated) literals that constitute the *body* (premise) of the rule. Commas in rules' bodies denote conjunction. Therefore, $\alpha \leftarrow \delta_1, \ldots, \delta_n$ is equivalent to $\alpha \vee$ not $(\delta_1 \wedge \ldots \wedge \delta_n)$. A rule with an empty body is denoted by $\alpha \leftarrow true$. Rules or literals are ground if they contain no variables. Following Prolog's convention, predicate and constant symbols start with a lower case letter, while variables start with a capital letter.

The Event Calculus (EC) [5] 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 [6]. 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 initiatedAt/2 and 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

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

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	
$\mathsf{holdsAt}(F, T+1) \leftarrow$ $\mathsf{initiatedAt}(F, T)$.	$\mathsf{holdsAt}(F, T+1) \leftarrow$ $\mathsf{holdsAt}(F, T)$, not $\mathsf{terminatedAt}(F, T)$.

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

as *walking*, or *standing still*. The second activity type corresponds to complex events complex events (long-term activities) and consists of activities that involve more than one person, e.g. two people *meeting each other*. The goal is to recognize complex events as combinations of simple events and additional domain 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 for each time-point.

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 $\mathcal{I}$, a background theory B, which in our case consists of the domain-independent axioms of the EC, and a 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 we restrict attention to stable models. Also, let $positives(H, I)$ (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

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

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

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

Table 2: **(a)** Example data from activity recognition. E.g., at time point 1 person 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 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 rule dictates that *moving* of 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 and their difference in direction is less than 45°.

where $\mathcal{L}(M)$ denotes the hypothesis language defined by the language bias M. The language bias that we assume is mode declarations [7], a form of user-defined directives that specify the signatures of predicates that may be used for constructing rules, thus restricting the language.

3. The OLED System

OLED [9] learns a theory by joining together independently-constructed rules, each of which is learnt in an online fashion. OLED relies on the Hoeffding bound [11] to approximate the quality of a rule on the entire input using only a subset of it. 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 rule in a top-down fashion, by specializing it using literals from a bottom rule [7]. The Hoeffding bound is utilized in the specialization process as follows: Given a rule evaluation function G and some rule r, OLED evaluates r and all of its candidate specializations on training examples that stream-in. Assume that after

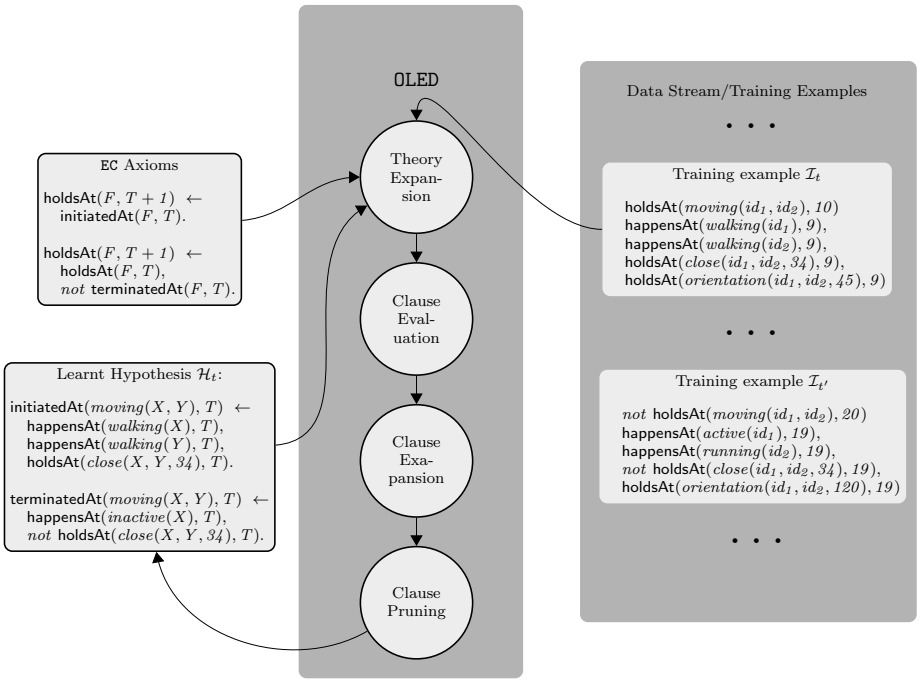

Figure 1: An overview of OLED's learning strategy.

n training examples from the input stream, 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 examples from the input stream until $\Delta\overline{G} > \epsilon$. These examples need not be stored or reprocessed. Each example is processed once to extract the necessary statistics for calculating G-scores and is subsequently discarded, thus giving rise to an online (single-pass) rule construction strategy. To ensure that no rule 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 hill-climbing strategy, where a single literal is added to a rule 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 rule length.

To calculate G-scores, each rule 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 data that stream-in. Also, r is equipped with an example counter that counts the number of examples (number of groundings of target complex

event instances) on which r has been evaluated so far and is used in the calculation of ϵ in the Hoeffding bound-based search heuristic. Although different scoring functions may be plugged into OLED, in this work we use precision, to score initiation rules, and recall, to score termination rules, as in [9]. Moreover, OLED supports a rule pruning mechanism, that allows to remove low-quality rules (e.g. rules that have been generated from noisy examples) and a tie-breaking mechanism, that allows to randomly select between equally good specializations. We refer to [9] for more details.

Figure 1 presents an overview of OLED. In the face of each training interpretation that streams-in, OLED goes through a series of actions. The first step is *theory expansion*, i.e. generation of new rules, which are subsequently added to the current theory. In this step, OLED initially detects potential examples (annotation atoms, such as holdsAt$(moving(id_1, id_2), 10)$) in the current training interpretation, which are not entailed by any of the existing rules in the theory. If such an example α exists, OLED starts growing a new rule that entails this example. To this end, it first constructs a new *bottom rule* [12, 7] from α, i.e a rule that contains in its body the maximum number of literals whose conjunction entails α. A bottom rule, denoted by $\perp$, is usually too restrictive to be of any use for event recognition and its role is to serve as a search space for subsets of its body conditions, which may define a high quality rule. Therefore, theory expansion consists of adding to the current theory a set of empty-bodied rules of the form $r_i = head(\perp_i) \leftarrow true$, where $\perp_i$ is a bottom rule. From that point on, each such rule r_i is gradually specialized by the addition of literals from $body(\perp_i)$, in a hill-climbing process, using Hoeffding tests, as previously described, to identify the best literal to add at each specialization step. The next step is *rule evaluation*, where all existing rules and their current candidate specializations are evaluated on the current interpretation. The *rule expansion* phase follows, where Hoeffding tests are performed for each existing rule, and those that pass the test are "expanded", i.e. replaced by their best-scoring specialization. The final step is the *rule pruning* phase, where low-quality rules are removed, and the loop continues with the next training interpretation.

4. Parallel Learning with OLED

We now proceed with the description of two different strategies for parallel learning with OLED. In both strategies, we assume that learning is performed by a set $\mathcal{N}$ of independent processing nodes, each handling a separate data stream, while using message passing to communicate with its peers. The goal is to learn a high-quality theory w.r.t. the union of the streams.

4.1. Synchronous Strategy

In our first parallel strategy, which we denote by sync, rules are learnt in a synchronous fashion, in the sense that all processing nodes contribute to the effort of learning each rule in a hypothesis. To this end, the rule evaluation process is performed by all nodes in parallel. When the Hoeffding test succeeds at some processing node, evaluation results from other nodes are combined in order to make a more informed decision. The pseudocode for the behavior of a single learning node in sync is presented in Algorithm 1. Upon receiving a new interpretation, sync follows the standard OLED learning

Algorithm 1 OLED-sync$(H, G, \mathcal{I}, \mathcal{N}')$

Input: H: An initial (potential empty) theory; G: A rule evaluation function; $\mathcal{I}$: A stream of training examples; $\mathcal{N}'$: Set of peer processing nodes.

Output: A learnt theory H.

1: **for each** $I \in \mathcal{I}$ **do**
2: **if** a rule r is received from some other node $N \in \mathcal{N}'$ **then**
3: $H \leftarrow H \cup r$.
4: $I_{inferred}$:= the interpretation inferred from $B \cup H \cup narrative(I)$
5: **for each** $\alpha \in annotation(I) \setminus I_{inferred}$ **do**
6: Generate a bottom rule $\bot$ from $B \cup I$, using α as a seed atom.
7: $r := head(\bot) \leftarrow true$.
8: $specializations(r) := \{head(r) \leftarrow body(r) \wedge \delta | \delta \in body(\bot)\}$.
9: $H \leftarrow H \cup r$.
10: Send $\{r\} \cup specializations(r)$ to each node $N \in \mathcal{N}'$.
11: **for each** $r \in H$ **do**
12: Compute the G-scores of each rule in $\{r\} \cup specializations(r)$ on I.
13: **for each** rule $r \in H$ **do**
14: **if** `HoeffdingTestSucceeds`(r) **then**
15: `PollAndAggregate`$(r, \mathcal{N}')$
16: **if** `HoeffdingTestSucceeds`(r) **then**
17: $r \leftarrow r'$, (r' is r's best specialization); Notify all $N \in \mathcal{N}$ to replace r by r'.
18: **if** `Prune`(r) **then**
19: `PollAndAggregate`$(r, \mathcal{N}')$
20: **if** `Prune`(r) **then**
21: $H \leftarrow H \setminus \{r\}$; Notify all $N \in \mathcal{N}'$ to also remove r.
22: **return** H
23:
24: **procedure** `PollAndAggregate`$(r, \mathcal{N}')$:
25: Poll each $N \in \mathcal{N}'$ for local counts for $r \cup specializations(r)$.
26: **if** replies received:
27: Update local TP, FP, FN counts for all rules in $r \cup specializations(r)$ by:
28: $TPs_r = TP_r + \sum_{N_j \in \mathcal{N}'} TP_r^j$; $FPs_r = FP_r + \sum_{N_j \in \mathcal{N}'} FN_r^j$; $FN_r = FN_r + \sum_{N_j \in \mathcal{N}'} FN_r^j$

strategy consisting of *theory expansion*, *rule evaluation*, *rule expansion* and *rule pruning*. However, it differs from the sequential OLED algorithm in the following respects:

Theory Expansion: When a node N_i generates a new rule r, it broadcasts r to all other nodes in $\mathcal{N}$ (line 10, Algorithm 1). Each node that receives such a message adds r to its own theory and it starts scoring r, and its candidate specializations, on its own data (lines 2 and 12, Algorithm 1). As in the single-core version of OLED, a new rule r consists of an empty-bodied rule $head(\bot_r) \leftarrow true$, where $\bot_r$ is a bottom rule generated at N_i.

Rule specialization: When a node N_i is about to specialize a rule r, i.e. when OLED's Hoeffding test for r, as described in Section 3, succeeds locally at N_i (line 14, Algo-

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.

Table 3: The main messages exchanged between data processing nodes in OLED's sync parallel learning strategy.

rithm 1), node N_i sends a message to all other nodes, requesting each node's evaluation statistics for rule r and its candidate specializations. Each recipient node responds by sending the requested statistics back to N_i. These statistics consist of TP, FP, FN and E counts for rule r and its candidate specializations, where by E we denote the number of examples on which a rule has been evaluated so far. The process is described in the `PollAndAggregate` procedure in Algorithm 1. The received counts for rule r and its specializations are combined with node N_i's local counts, as shown in line 28 of Algorithm 1.

Each processing node $N_i \in \mathcal{N}$ maintains a record, for each rule 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 rule r from node $N_j, j \neq i$, 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 rule evaluation statistics are combined as described above, node N_i repeats the Hoeffding test for rule r to assess if the test still succeeds after the accumulated counts from all other nodes, for rule r and its specializations, have been taken into account. If it does, 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 message to all other nodes, instructing them to also replace r 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, rule r is not specialized and all nodes continue evaluating their theories on new incoming examples from their training streams.

Rule pruning: For a rule r to be pruned away, two conditions must hold: First, 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, observed so far in the learning process, 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 rule specializations so far. Second, from that point on where r remains unchanged, a sufficiently large number of examples must be seen, in order to use a Hoeffding test to infer that, with probability $1 - \delta$, the quality of rule r is below a user-defined pruning threshold.

In the parallel version of OLED, each node uses the above heuristics to decide locally whether a rule r should be pruned. Once it has seen enough data from its own stream to make that decision for r, it requests from all other nodes the necessary statistics for r, which node N_i uses to re-assess whether r should be pruned, based on the global view obtained by combining r's separate evaluations from all processing nodes. If node N_i eventually decides to prune r, it notifies all other nodes to also remove rule r from their theories. This is presented in lines 18-21 of Algorithm 1. Table 3 summarizes the various types of message that learning nodes exchange in the sync learning strategy.

The following proposition imposes an upper bound on the number of messages exchanged by processing nodes in sync.

Proposition 1. *sync needs up to $\mathcal{O}(N^2 k|literals_H|)$ messages at the worst case, to construct a theory H using N processing nodes, where $|literals_H|$ denotes the total number of literals in H and k is some measure of noise, either inherent in the training data, or introduced by not processing the data sequentially.*

Proof Let H be a theory returned by sync at any time during the synchronous distributed learning process. Let us assume for simplicity that the default specialization strategy is followed, i.e. specializations are implemented by adding a single literal to the body of a rule at each specialization step. Up to the point where sync has constructed H, each node N_i may have sent up to $2k(N-1)|literals_H|$ messages to the remaining $N-1$ nodes: For each literal in a rule (i.e. for each specialization step), N_i sends two $2+m$ messages, namely, one message to request statistics for the specialization, one to notify its peers in case the specialization was actually implemented, plus m messages, requesting statistics for specializations that were not implemented after the accumulated counts were taken into account). We therefore have that N_i sends

$$(2+m)(N-1)|literals_H| \leq 2kN|literals_H|$$

where in the formula above we approximate m by k. This is because the specialization attempts of a single node that were actually "canceled" after the accumulated statistics from all nodes were taken into account, may be attributed to the "myopia" induced by

Algorithm 2 d-OLED-async$(H, G, S_{min}\mathcal{I}, \mathcal{N}')$

Input: H: An initial (potentially empty) hypothesis; G: A rule evaluation function; $\mathcal{I}$: A stream of training examples; $\mathcal{N}'$: Set of peer processing nodes.

Output: A learnt hypothesis H.

1: **for each** $I \in \mathcal{I}$ **do**
2: **if** a rule r is received from some other node in $\mathcal{N}'$ **then**
3: **if** $r \notin H$ **then**
4: $H \leftarrow H \cup \{r\}$.
5: **else**
6: Aggregate the statistics of the received rule r to those of the local copy of r.
7: "Flag" r to avoid re-sending it.
8: $H \leftarrow \mathsf{OLED}(H, G, I)$
9: **for** each $r \in H$ **do**
10: **if** `IsStable`(r) and $score(r) > S_{min}$ **then**
11: Send r to each node in $\mathcal{N}'$.
12: **Return** H

not seeing the data sequentially, which we assume that is quantified by k. We need to add to a number of messages related to rule pruning. Since a rule that turns out to be of low quality and needs to be pruned is generated from a noisy example, we approximate the average number p of such bad rules by k. With reasoning similar as before, it can be seen that the communication overhead related to pruning is bound by $2Nk$, we therefore have that the total number of messages sent by a single node N_i is

$$2Nk|literals_H| + 2Nk = \mathcal{O}(Nk|literals_H|)$$

Multiplying by N (for all N nodes) yields the result.

■

4.2. Asynchronous Strategy

We propose an alternative, asynchronous learning strategy, which we henceforth denote by async. In this strategy rules are learnt independently at each node and the best rules from each node are made available to its peers. This strategy requires a significantly smaller amount of communication between processing nodes than the sync strategy. The price is that the learning process converges to a good hypothesis at a slower rate, as compared to the synchronous strategy. The pseudocode for the behaviour of a single learning node in the async strategy is presented in Algorithm 2.

In async, Each node $N_i \in \mathcal{N}$ uses the monolithic OLED algorithm to learn a set of rules from its local training stream (line 8, Algorithm 2). During this process, a node N_i may identify a locally "good" rule. Intuitively, a good rule is one that over time has converged to a "stable" version of adequate quality (line 10, Algorithm 2), meaning that it has not changed (been specialized) for a sufficiently large amount of time and its score so far exceeds a given threshold. Once a node N_i identifies a high-quality rule r, it broadcasts r to all other nodes (line 11, Algorithm 2), as a potentially useful rule that may enrich the local theories of N_i's peer nodes. The recipient nodes add r to their local theories, in case they have not already discovered r themselves, i.e. they do

not have r in their local theories (lines 2–4). From that point on, each recipient node N_j treats rule r as part of its own theory, meaning that r starts being evaluated on the local stream of N_j and it may be further specialized, if this improves its quality, or it may even be pruned away, in case its performance on the local training stream S_j decreases substantially. If some node receives a rule r that already exists in its local theory, it simply adds the evaluation statistics of the received rule to those of its local copy of r and "flags" r to avoid re-sending it over the network as a newly-discovered, high-quality rule.

In order for an individual node N_i to decide when one of its local rules r is stable (good enough to be shared with its peer nodes), async uses a heuristic "dual" to the one used for rule pruning (discussed in Section 4.1). A rule is considered stable if it has not been changed (specialized) for a period set to the average value of $n = \mathcal{O}(\frac{1}{\epsilon^2} ln \frac{1}{\delta})$ that has resulted in rule specializations so far; and it is considered of sufficient quality if its score from that point on exceeds a given rule quality threshold for a sufficiently large number of examples.

As is the case with the single-core algorithm and its synchronous parallel version discussed in the previous section, async is an any-time learner, i.e. it may output a theory at any time during the learning process. To do so, async selects from each node individual rules that are good enough for most of the other processing nodes, i.e. rules that are found in the local theories of the majority of the processing nodes. To ensure that, each selected rule should be stable w.r.t. the majority of the nodes, meaning that it should have been sufficiently evaluated and its quality during this evaluation does not drop significantly to justify it being further specialized or pruned. The behaviour of a single learning node in the async strategy is presented in Algorithm 2. The following proposition provides an upper bound for the communication overhead of the async strategy.

Proposition 2. *async needs $\mathcal{O}(N^2(k + |rules_H|))$ messages to generate a theory H, where N and k are as in Proposition 1 and $|rules_H|$ denotes the number of rules in H.*

Proof Let H be a theory returned by async, so that H consists of the high-quality, stable rules found in the local theories of the majority of the N processing nodes. Each node N_i may have sent at most $(N-1)(k+|rules_H|)$ messages to the rest of the $N-1$ nodes: At most $|rules_H|$-many messages (one for each rule in H), plus a number of messages (which we approximate by the noise parameter k), for communicating rules that seemed good locally at N_i, but turned out to be of low quality globally, for the majority of processing nodes. We therefore have that the communication overhead of a single rule is at most

$$(N-1)(k + |rules_H|) < N(k + |rules_H|) = \mathcal{O}(N(k + |rules_H|))$$

and multiplying by N (for the total communication cost) we get the upper bound. ■

Propositions 1 and 2 indicate that the maximum communication overhead of the async strategy is smaller than that of sync assuming that the number of rules in a theory is typically much smaller than the total number of literals in the theory. The tradeoff is that async is expected to converge to good hypotheses at a slower rate in the general

	#Cores	Time (sec)		Speed-up		F_1-score		Theory size		# Msgs	
(A) *meet*	1	46 (±7.34)		–		0.798		**28** (±3.28)		–	
		sync	async	sync	async	sync	async	sync	async	sync	async
	2	18 (±4.23)	32(±3.12)	2.5	1.4	**0.818**	0.809	31 (±5.12)	33 (±4.43)	78 (±18.32)	**22** (±12.7)
	4	15 (±3.88)	23 (±4.04)	3	2	0.805	0.811	34 (±5.62)	34 (±5.14)	133 (±22.18)	62 (±8.44)
	8	15 (±2.46)	21 (±1.34)	3	2.1	0.802	0.811	35 (±4.76)	35 (±5.02)	360 (±31.24)	112 (±21.45)
	16	**14** (±1.08)	22 (±1.78)	**3.2**	2	0.8	0.8	35 (±5.19)	35 (±4.77)	684 (±38.34)	187 (±22.76)
move	1	68 (±11.23)		–		**0.744**		**21** (±6.23)		–	
		sync	async	sync	async	sync	async	sync	async	sync	async
	2	31 (±8.12)	51 (±12.45)	2.1	1.3	0.740	0.743	**21** (±5.43)	**21** (±3.6)	**56** (±13.76)	**18** (±4.18)
	4	27 (±5.22)	42 (±9.44)	2.5	1.6	0.739	0.741	**21** (±4.92)	**21** (±3.62)	121 (±24.23)	58 (±12.4)
	8	**26** (±6.14)	40 (±13.51)	**2.6**	1.7	0.743	**0.744**	23 (±3.88)	**21** (±4.72)	278 (±21.43)	104 (±9.63)
	16	**26** (±6.23)	44 (±7.15)	**2.6**	1.5	0.730	0.741	23 (±3.64)	23 (±3.22)	590 (±48.15)	166 (±14.17)
(B) *meet*	1	7588 (±162.67)		–		**0.834**		**36** (±2.22)		–	
		sync	async	sync	async	sync	async	sync	async	sync	async
	2	2144 (±177.65)	2132 (±198.43)	3.5	3.5	**0.834**	0.832	**36** (±1.12)	**36** (±0.87)	**82** (±7.48)	**22** (±5.61)
	4	1682 (±58.24)	1672 (±52.13)	4.5	4.5	**0.834**	0.831	**36** (±1.78)	**36** (±1.23)	143 (±16.48)	66 (±7.56)
	8	912 (±34.76)	883 (±42.76)	8.3	8.5	0.832	**0.834**	**36** (±1.45)	**36** (±1.17)	368 (±28.54)	108 (±12.02)
	16	**623** (±31.72)	**582** (±18.12)	**12.1**	**13**	0.832	0.832	**36** (±2.04)	**36** (±1.83)	692 (±33.48)	182 (±12.28)
move	1	7898 (±233.3)		–		**0.758**		**34** (±1.08)		–	
		sync	async	sync	async	sync	async	sync	async	sync	async
	2	2312 (±83.27)	2276 (±125.46)	3.4	3.4	0.753	0.752	**34** (±1.92)	**34** (±1.27)	55 (±11.45)	**20** (±7.2)
	4	1788 (±113.08)	1761 (±72.2)	4.4	4.4	0.756	**0.758**	**34** (±2.87)	**34** (±1.25)	122 (±18.43)	61 (±16.77)
	8	966 (±73.48)	932 (±108.12)	8.1	8.4	0.753	0.754	**34** (±1.52)	**34** (±1.12)	284 (±27.22)	105 (±10.6)
	16	681 (±82.34)	**634** (±43.8)	11.5	**12.4**	0.754	0.749	**34** (±1.07)	**34** (±1.23)	602 (±34.21)	181 (±15.24)

Table 4: Experimental results on the CAVIAR dataset.

case. This is because each node in the async strategy learns rules independently, while in sync nodes collaborate, processing in parallel larger amounts of data in the same amount of time. Additionally, the async strategy requires extra time, after a locally good rule is found, in order to evaluate the rule on all other nodes.

5. Experimental Evaluation

We present an experimental evaluation of the proposed parallel learning strategies and a comparison with the sequential OLED algorithm. We use three real datasets from the domains of activity recognition and maritime surveillance. All experiments were conducted on a Linux machine with a 1.2GHz processor (8 cores and 16 threads) and 251Gb of RAM. The algorithms are implemented in the Scala programming language. They use the Clingo[2] answer set solver for logical reasoning and Scala's akka Actors library[3] to model node behavior. The code and data are available online[4].

[2] http://potassco.sourceforge.net/

[3] http://akka.io/

[4] https://github.com/nkatzz/OLED

5.1. Activity Recognition

For our first set of experiments we used CAVIAR (described in Section 2), a benchmark dataset for human activity recognition, consisting of 282,067 training examples. We performed learning with 1, 2, 4, 8 and 16 processing cores for constructing the definitions of two target complex events, related to two persons *meeting each other* or *moving together*. CAVIAR contains 6,272 positive examples for *moving* and 3,722 for *meeting*. For the experiments with the parallel versions of OLED, positive and negative examples for the target complex events were evenly distributed across different processing cores. The results were obtained by ten-fold cross-validation and are presented in Table 4 in the form of averages for training time, F_1-score and theory size, as well as average exchanged message number. Due to the fact that testing sets were highly imbalanced, since positive examples form sequences of uneven length, F_1-scores were obtained by micro-averaging results from each fold. In all experiments the training data were presented to the learners in mini-batches, each containing ten CAVIAR video frames. Table 4 also presents the achieved speed-ups, defined as $\frac{T_1}{T_n}$, where T_1 and T_n are respectively the training times of a sequential 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 less (resp. greater) than n.

Table 4 is divided into two sub-tables, (A) and (B) respectively. Table (A) presents results from the regular CAVIAR dataset, while Table (B) presents results from a version of CAVIAR consisting of "larger" examples. It consists of 10 copies of the original dataset, where each copy differs from the others only in the constants referring to the tracked entities (persons, objects) in simple and complex events. This dataset contains 100 different tracked entities, as compared to only 10 entities of the original CAVIAR dataset. Each extra entity introduced to the dataset has the same "behavior" in time, in terms of low-level activities, as one of the original tracked entities in CAVIAR. However, the (x, y)-coordinates of each extra entity p' have been changed, to ensure that it does not coincide spatially with the original tracked entity p, whose behavior in time is copied by p'. Also, the annotation in the new dataset, in terms of high-level activities (*meeting* and *moving*) was also enriched, to take into account the extra entities introduced in the dataset. The experimental setting for the CAVIAR-(B) case was as described above.

Starting from the CAVIAR-(A) experiment, we see that parallel learning resulted in theories of slightly higher F_1-score for *meeting*, as compared to single-core learning. In the sequential setting, OLED postpones the generation of new rules, up to the point where existing rules become too specific to cover new examples. During this time, positive examples which may result in good rules (recall that OLED learns by "encoding" examples into bottom rules), are "skipped", i.e. they are not used for learning new rules, since they are covered by existing ones. In contrast, the data distribution in both of the parallel settings resulted in cases where interesting examples that would have been missed in the sequential setting, are actually used for learning. This resulted in OLED learning slightly "richer" theories for *meeting* in both of the parallel settings. A similar effect was not observed for *moving*, which has a simpler definition than *meeting*.

To obtain comparable F_1-scores between the sync and the async parallel learning strategies, we had to allow for larger training times for the latter strategy. A preliminary

test run with *meeting* as the target concept, indicated that a single pass over each processing node's training set was inadequate for async to produce high-quality theories. The reason is that, while async was in principle capable to learn good rules, additional training time was required, in most cases, for evaluating such rules on all nodes long enough to infer that they are stable. Such additional training time was not always available with a single pass over the data (nodes exhausted their training sets too early and learning was terminated). As a result, although good rules were discovered, many of them were excluded from the final hypothesis on the grounds of not yet being stable at the majority of the processing nodes. In a test experiment with a single pass over the data, async produced theories with an average F_1-score of 0.664 for *meeting*, as opposed to sync's 0.807 score. To obtain the results reported in Table 4(A), we therefore allowed each node in async to iterate twice over its training set. This is an effect of the slower convergence rate of the async strategy and it is reflected in its higher training times in Table 4(A), as compared to the training times of the sync strategy. On the other hand, as was expected, async requires a significantly smaller amount of communication, in terms of exchanged messages.

Both parallel strategies in the CAVIAR-(A) experiment have a sub-linear speed-up pattern. This is not the case however in the CAVIAR-(B) experiment, where both strategies achieve super-linear speed-ups. This seems to imply that the gain in efficiency of our proposed parallel learning strategies increases with the difficulty of the learning task at hand, in terms of the "unit cost" of processing individual examples. Indeed, the CAVIAR-(B) dataset consists of "larger" training examples, each containing an increased number of ground domain literals and domain constants. Such examples are significantly harder to be reasoned upon, as indicated by the the exponential growth in training times for all learners in the CAVIAR-(B) experiment.

It is worth noting that increasing the size of each training example in the CAVIAR-(B) experiment affects the performance of the async algorithm in the following respect: The number of examples on which rules are evaluated in the core online learning strategy of OLED (the n in the Hoeffding bound test) is actually the number of groundings of target predicates (complex event instances). As the number of constants in the CAVIAR-(B) dataset increases, so does the number of such groundings, and as a result, the count of examples on which rules are evaluated. This makes redundant the extra training time (devoted mostly on rule evaluation) that was necessary for the async strategy to produce high-quality theories in the CAVIAR-(A) experiment. Indeed, in the larger CAVIAR-(B) dataset, async learns good theories with each node passing once over its training set, and it is actually slightly faster than the sync strategy. This may be attributed to the increased communication overhead of the sync strategy.

Due to the increase in training data size in the CAVIAR-(B) experiment, F_1-scores for all learners are improved as compared to the CAVIAR-(A) experiment. In the CAVIAR-(A) experiment, good rules were often constructed "too-late", from examples that were encountered shortly before the data were exhausted. Such rules may be discarded, since OLED (and its parallel versions) uses a "warm-up" period parameter that controls a minimum number of examples on which a rule must be evaluated in order to be included in an output hypothesis. In contrast, in the CAVIAR-(B) experiment such problems were avoided, thanks to the increase in training data size.

5.2. Maritime Surveillance

For our second experiment we used a real, publicly available dataset from the field of maritime monitoring[5]. The dataset consists of Automatic Identification System (AIS) position signals collected from vessels sailing in the area of Brest, France, between 1 October and 31 March 2015. The data have been pre-processed using trajectory compression techniques [13], whereby major changes along each vessel's movement are tracked. This process can instantly identify *critical points* along each trajectory, such as a stop, a turn, or slow motion. Using the retained movement features, i.e. the critical points, we may reconstruct the trajectory of a vessel with small acceptable deviations from the original one. The dataset was additionally pre-processed in order to extract spatial relations between vessels and areas of interest, such as protected areas, and between pairs of vessels.

A number of complex events have been defined, in collaboration with the domain experts of the datAcron project[6], such as whether a vessel moves dangerously fast in a designated area, whether two vessels meet in the open sea and so on [14]. The dataset concerns 4,961 vessels and 6,894 areas amounting to approx. 1.3 GB.

In order to obtain annotation, in terms of target complex event instances, we used hand-crafted complex event definitions, developed in datAcron, to perform inference on the maritime data. We experimented with three complex events: *rendezvous*, which holds when two vessels are stopped, or move with low speed in proximity to each other in the open sea; *loitering*, which holds when a vessel remains idle in the open sea; and *highSpeedIn*, which holds when a vessel is moving within an area with a speed that exceeds the area's a speed limit. The definition of the *highSpeedIn* complex event, for example, is the following::

$$
\begin{aligned}
&\mathsf{initiatedAt}(highSpeedIn(Vessel, AreaName), T) \leftarrow \\
&\quad \mathsf{happensAt}(isInArea(Vessel, AreaName), T), \\
&\quad \mathsf{happensAt}(velocity(Vessel, Speed), T), \\
&\quad speedLimit(AreaName, SpeedArea), \\
&\quad Speed > SpeedArea.
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{terminatedAt}(highSpeedIn(Vessel, AreaName), T) \leftarrow \\
&\quad \mathsf{happensAt}(isInArea(Vessel, AreaName), T), \\
&\quad \mathsf{happensAt}(velocity(Vessel, Speed), T), \\
&\quad speedLimit(AreaName, SpeedArea), \\
&\quad Speed \leq SpeedArea.
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{terminatedAt}(highSpeedIn(Vessel, AreaName), T) \leftarrow \\
&\quad \mathsf{happensAt}(leavesArea(Vessel, AreaName), T).
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{terminatedAt}(highSpeedIn(Vessel, AreaName), T) \leftarrow \\
&\quad \mathsf{happensAt}(gap\_start(Vessel), T).
\end{aligned}
$$

The first rule states that $highSpeedIn(Vessel, Area)$ is initiated at time T if the $Vessel$

[5] https://zenodo.org/record/1167595#.WzOOGJ99LJ9
[6] http://datacron-project.eu/

	#Cores	Time (sec)		Speed-up		F_1-score		Theory size		# Msgs	
Rendezvous	1	4688 (±213.65)		–		**0.923**		**18** (±0.78)		–	
		sync	async	sync	async	sync	async	sync	async	sync	async
	2	1234 (±112.02)	977 (±107.5)	3.7	4.8	0.918	0.921	19 (±2.04)	**18** (±1.34)	**54** (±8.23)	**12** (±4.12)
	4	989 (±134.18)	923 (±98.45)	4.7	5.07	0.921	0.921	19 (±0.77)	19 (±0.8)	148 (±17.54)	33 (±9.08)
	8	686 (±75.87)	622 (±98.23)	6.8	7.8	0.922	0.921	20 (±1.23)	19 (±1.44)	324 (±15.34)	77 (±8.97)
	16	558 (±63.08)	**492** (±72.14)	8.4	**9.5**	0.921	0.921	20 (±2.02)	20 (±1.88)	786 (±24.5)	162 (±18.09)
Loitering	1	1282 (±74.12)		–	–	**0.944**		**16** (±1.55)		–	–
		sync	async	sync	async	sync	async	sync	async	sync	async
	2	586 (±42.23)	512 (±32.46)	2.1	2.5	0.938	0.932	19 (±2.14)	**16** (±0.8)	62 (±12.09)	**22** (±5.12)
	4	381 (±28.56)	307 (±32.28)	3.3	4.1	0.941	0.939	19 (±2.43)	19 (±1.23)	188 (±17.34)	58 (±8.22)
	8	281 (±22.18)	198 (±24.65)	4.5	6.5	0.942	0.941	21 (±1.92)	19 (±0.98)	402 (±21.34)	95 (±8.65)
	16	**204** (±25.23)	**177** (±17.07)	**6.2**	**7.2**	0.932	0.941	20 (±2.48)	19 (±1.87)	865 (±19.34)	203 (±11.07)
HighSpeedIn	1	5112 (±207.34)		–		**0.986**		**16** (±1.12)		–	
		sync	async	sync	async	sync	async	sync	async	sync	async
	2	1284 (±112.2)	1034 (±184.53)	3.9	4.9	0.984	0.984	18 (±1.32)	18 (±1.04)	78 (±6.4)	**21** (±3.08)
	4	885 (±88.25)	812 (±109.23)	5.7	6.2	0.984	0.984	18 (±1.14)	18 (±1.34)	202 (±11.82)	66 (±7.12)
	8	714 (±64.28)	688 (±113.7)	7.1	7.4	0.984	0.984	25 (±2.02)	25 (±1.97)	487 (±28.44)	198 (±12.34)
	16	**478** (±44.45)	**458** (±62.8)	**10.7**	**11.1**	0.984	0.984	18 (±1.43)	18 (±1.56)	1086 (±34.87)	402 (±14.04)

Table 5: Experimental results on the Brest maritime dataset.

is within the $Area$ at T and its speed exceeds the limit of the $Area$. The second rule states that $highSpeedIn(Vessel, Area)$ is terminated at time T if the $Vessel$'s speed complies with the limit. The last two rules state respectively that $highSpeedIn(Vessel, Area)$ is terminated at T if the $Vessel$ leaves the $Area$ at T, or if there is a gap in the communication with the $Vessel$ at T.

Partitioning the dataset geographically for experimenting with the parallel algorithms resulted in highly unbalanced partitions, with the vast amount of vessel activity taking place in just one or two of the resulting partitions, due to the concentration of vessel activity in ports. We therefore partitioned the data based on vessel ids, with each processing core handling a subset of the training data concerning particular vessels. Since the *RendezVous* complex event involves pairs of vessels, vessel ids which appear together in positive instances of this complex event were assigned to the same processing node.

As in the previous experiments, we performed a tenfold cross-validation experiment using 2, 4, 8 and 16 processing cores for the parallel versions of OLED and measuring training times, F_1 scores, number of exchanged messages and theory sizes. The results are presented in Table 5. Concerning F_1-scores, all learners produced theories of reasonable quality for all complex events. Speed-ups follow a sub-linear pattern for *loitering*, but they are much higher for *rendezVous* and *highSpeedIn* complex events. This supports our previous observation that the gain in efficiency of the parallel learning strategies increases with the difficulty of the learning task. Indeed, both the *rendezVous* and the *highSpeedIn* complex events are relational, the former involving pairs of vessels and the latter vessel-area pairs, in contrast to the *loitering* complex event which only involves a single vessel. Moreover, each training interpretation contains a large number of vessel and area constants, which are hard to be reasoned upon, especially for the relational target events, as the increased training times for *rendezVous* and *highSpeedIn* indicate. Regarding (communication) efficiency, the results indicate that the

	#Cores	Time (sec)		Speed-up		F_1-score		Theory size		# Msgs	
Rendezvous	1	9413 (±205.77)		–		**0.914**		**24** (±1.12)		–	
		sync	async	sync	async	sync	async	sync	async	sync	async
	2	3225 (±174.88)	3118 (±201.56)	2.9	3.01	0.901	0.897	**24** (±1.87)	**24** (±1.89)	66 (±12.23)	**16** (±4.76)
	4	2125 (±166.43)	1848 (±99.08)	4.45	5.09	**0.914**	0.916	25 (±1.124)	**24** (±1.3)	183 (±21.12)	45 (±5.8)
	8	1762 (±102.07)	1222 (±87.34)	5.3	7.7	**0.914**	**0.914**	25 (±1.87)	25 (±1.102)	367 (±21.13)	88 (±7.76)
	16	1112 (±104.34)	**988** (±74.65)	8.46	**9.52**	0.911	0.913	31 (±2.43)	25 (±1.15)	884 (±22.44)	192 (±8.16)
Loitering	1	2488 (±112.45)		–	–	**0.928**		**20** (±0.8)		–	–
		sync	async	sync	async	sync	async	sync	async	sync	async
	2	1123 (±107.12)	1015 (±77.88)	2.2	2.4	0.922	0.919	22 (±1.64)	22 (±1.75)	78 (±11.12)	**18** (±6.23)
	4	673 (±74.12)	645 (±43.12)	3.6	3.8	0.917	0.925	22 (±1.18)	23 (±1.12)	172 (±13.8)	34 (±5.09)
	8	418 (±67.09)	373 (±29.67)	5.9	6.6	0.924	0.925	22 (±1.43)	22 (±1.02)	343 (±13.35)	79 (±7.76)
	16	323 (±32.4)	**284** (±21.7)	7.7	**8.7**	0.921	0.924	22 (±1.66)	23 (±1.05)	723 (±24.3)	198 (±8.23)

Table 6: Experimental results on the Mediterranean maritime dataset.

async strategy is significantly faster and requires less communication than the sync strategy.

To stress-test further the proposed learning strategies, we performed an additional experiment with a more challenging dataset from the datAcron project. The dataset consists of AIS position signals from 26,756 vessels, collected throughout the Mediterranean sea within a period of one week week in January 2016. Position signals from both terrestrial and satellite receivers are included in the dataset, which amounts to approx. 1.5 GB of events. Due to the much larger number of vessels involved in the Mediterranean dataset (recall that the Brest dataset involves only 4,961 vessels), the Mediterranean dataset is significantly harder to reason with. As in the previous maritime experiment, the data have been pre-processed to extract critical points related to vessels' mobility, in addition to spatial relations, such as vessel proximity. We performed experiments aiming to learn the definitions of two complex events, namely *rendezVous* and *loitering*. As in the previous setting, we used tenfold cross-validation with 1, 2, 4, 8 and 16 processing cores, while the dataset was partitioned based on vessel id.

Table 6 presents the results. All learners produce theories of comparable predictive accuracy. Regarding training times, we observe that they are elevated, as compared to the Brest dataset, due to the increased number of domain constants (vessel ids) in the Mediterranean sea. This is particularly so for the *rendezVous* complex event, which is relational. Notably, both parallel learners achieve significant speed-ups for both complex events, with the async strategy being better. Furthermore, similar to the previous results, async requires significantly less communication than the sync strategy.

6. Related Work

Machine learning techniques are attracting attention in the Complex Event Processing community [15, 16, 17]. However, existing approaches exhibit several limitations [18]: they often resort to ad-hoc learning techniques, which are hard to evaluate in more generic learning settings, while they have limited support for background knowledge and assume a batch learning setting. In contrast, we adopt a framework with direct

routes to well-established relational learning techniques [7], which can overcome such limitations. Moreover, using the Event Calculus allows for efficient event recognition via dedicated reasoners, such as RTEC [6].

Online learning settings, as the one we assume in this work, are under-explored in Inductive Logic Programming (ILP), where only a few approaches have been proposed. In [19] the authors present an online algorithm generating a rule from each miss-classified example in a stream, aiming to construct a theory that accounts for all training examples. In [20] the authors propose an online learner based on Aleph[7] and Winnow [21]. The algorithm maintains a set of rules, representing the "relational features" of the domain, which are combined to classify new incoming examples. As in Winnow, the weights of rules are updated via a mistake-driven update scheme. New rules are greedily generated by Aleph from miss-classified examples. A similar approach is put forth by OSL [22], an online learner for Markov Logic Networks (MLN) [23], and OSLα [24], an extension of OSL tailored towards learning with the Event Calculus, via exploiting properties of the background knowledge. OSL greedily generates new rules to account for miss-classified examples that stream in, and then relies on weight learning to identify which of these rules are relevant (the weights of irrelevant rules are eventually zeroed). More recently, OLED's online learning strategy has also been adapted to an MLN setting [25], using the AdaGrad algorithm to learn weights in the MLN semantics for the rules that OLED constructs. Other approaches to streaming relational learning are oriented towards unsupervised tasks like frequent pattern discovery [26], or rely on propositionalization techniques and off-the-self, online propositional learners[27]. A key difference between our work and these approaches is that we proposed techniques for *parallel* learning, consuming disjoint data streams to construct globally optimal theories.

A preliminary version of our proposed approach was presented in [28]. We extend significantly [28] via introducing the async parallel learning strategy and evaluating both sync and async on two additional real, challenging datasets from the field of maritime surveillance.

Regarding parallel relational learning in general, a substantial amount of work exists in the ILP literature. Thorough reviews may be found in [29, 30]. Such ILP algorithms exploit parallelism across three main axes [29]: Searching through the hypothesis space in parallel (search parallelism); splitting the training data and learning from data subsets (data parallelism); and evaluating candidate rules in parallel (evaluation/coverage parallelism).

In [31] 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 rules 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 [31] learns much faster than a sequential algorithm, achieving super-linear speed-ups. A similar approach is proposed in [32], where the training examples are split across multiple nodes and searched in parallel, while the best rules from each node are "pipe-lined" to all other nodes.

[7] http://www.cs.ox.ac.uk/activities/machinelearning/Aleph/aleph

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 rules are performed in parallel, on disjoint partitions of the data. In the latter case, bottom clauses are generated and searched in a concurrent fashion from several "seed" examples. The reducer then selects the best rule that results from this process. A similar approach for parallel exploration of independent hypotheses has been proposed in [33], while approaches towards parallel coverage tests have been proposed in [34, 35]. In [36], the approach of [30] was extended to a framework that is capable of automatically regulating the learning effort across multiple nodes. Finally, work on distributed logical learning from a multi-agent systems perspective has been reported in [37, 38]. A main difference of the work presented here from the aforementioned approaches to parallel ILP is that they rely on (offline) iterative algorithms, which require several passes over the data to compute a hypothesis. In contrast, OLED is an online, single-pass algorithm.

Regarding learning outside ILP, Vu et al. [39] present AMRules (Adaptive Model Rules), an online framework for the parallel construction of regression rules. The rule learning strategy in AMRules is similar to OLED's: Rules are gradually expanded by adding features to their bodies, accumulating statistics for each feature from the training data and using Hoeffding tests to decide which feature to add at each such expansion step. The parallel AMRules algorithm assumes a number of learner nodes and relies on a vertically parallel scheme, where each learner node handles a separate fragment of the global rule set. The entire rule set in maintained at a designated "model aggregator" node, which performs coverage tests on each incoming training example and identifies the rules that fire on this example. The example is then broadcast to the learner node that is responsible for handling each such firing rule. The learner node updates the rule's statistics from the received training example, performs a Hoeffding test and if the test succeeds, resulting to a rule expansion, it sends the expanded rule to the model aggregator. The latter is also responsible for generating new rules by expanding an empty-bodied rule using examples which are not covered by any of the existing rules. When a new rule is generated, the model aggregator sends it to a learner node, which, from that point on, is responsible for handling that rule. As noted in [39], the centralized approach that relies on a single model aggregator is a bottleneck that results in low throughput. To address the issue, the authors in [39] propose an alternative algorithm that uses both horizontal and vertical parallelism. In this hybrid version of parallel AMRules, the model aggregator is replicated in multiple copies, each of which maintains the entire rule set but handles a separate fragment of the input (horizontal parallelism), while the learner nodes use the vertical parallelism scheme described above. In the hybrid version of parallel AMRules, the generation of new rules remains centralized and is handled by a designated node which receives training examples that are not covered by an existing rule.

There are several differences between parallel AMRules and the parallel OLED strategies presented in this work. First, AMRules is designed for regression tasks, while OLED is designed for classification. Second, AMRules is strictly propositional, while OLED can naturally handle relational phenomena, such as vessel rendez-vous. Third, the parallel AMRules algorithm involves moving data around the processing nodes (recall that model aggregator nodes forward training examples to learner nodes), in ad-

dition to rules which are communicated between learners and model aggregators. In contrast, in both parallel learning strategies for OLED, communication is restricted to the exchange of rules, to avoid the communication overhead of exchanging data, which may become a bottleneck in distributed settings. Fourth, OLED's parallel strategies are fully decentralized, in contrast to parallel AMRules, which assumes centralized model aggregation in its vertically-parallel version, or centralized new rule generation, in its hybrid version that combines horizontal and vertical parallelism.

7. Summary and Future Work

We presented two parallel learning extensions of an online algorithm for the automatic construction of complex event definitions from relational data streams. Furthermore, we presented an experimental evaluation of our approach on a real, benchmark dataset from the domain of human activity recognition, and two real datasets from the field of maritime monitoring covering wide geographical areas, and thousands of vessels and areas of interest. Our results demonstrate that our proposed parallel learning strategies can significantly reduce training times, while they are capable of super-linear speed-ups on some occasions, without compromising the quality of the learnt hypotheses. For future work, we aim to evaluate and potentially extend these approaches in the distributed setting.

Acknowledgments.

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

References

[1] O. Etzion, P. Niblett, Event processing in action, Manning Publications Co., 2010.

[2] G. Cugola, A. Margara, Processing flows of information: From data stream to complex event processing, ACM Comput. Surv. 44 (3) (2012) 15:1–15:62.

[3] E. Alevizos, A. Skarlatidis, A. Artikis, G. Paliouras, Probabilistic complex event recognition: A survey, ACM Comput. Surv. 50 (5) (2017) 71:1–71:31.

[4] A. Artikis, A. Skarlatidis, F. Portet, G. Paliouras, Logic-based event recognition, Knowledge Eng. Review 27 (4) (2012) 469–506.

[5] R. Kowalski, M. Sergot, A logic-based calculus of events, New Generation Computing 4 (1) (1986) 67–95.

[6] A. Artikis, M. Sergot, G. Paliouras, An event calculus for event recognition, Knowledge and Data Engineering, IEEE Transactions on 27 (4) (2015) 895–908.

[7] L. De Raedt, Logical and relational learning, Springer Science & Business Media, 2008.

[8] J. Gama, Knowledge discovery from data streams, CRC Press, 2010.

[9] N. Katzouris, A. Artikis, G. Paliouras, Online learning of event definitions, TPLP 16 (5-6) (2016) 817–833.

[10] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer set solving in practice, Synthesis Lectures on Artificial Intelligence and Machine Learning 6 (3) (2012) 1–238.

[11] W. Hoeffding, Probability inequalities for sums of bounded random variables, Journal of the American statistical association 58 (301) (1963) 13–30.

[12] S. Muggleton, Inverse entailment and progol, New generation computing 13 (3) (1995) 245–286.

[13] K. Patroumpas, E. Alevizos, A. Artikis, M. Vodas, N. Pelekis, Y. Theodoridis, Online event recognition from moving vessel trajectories, GeoInformatica 21 (2) (2017) 389–427.

[14] M. Pitsikalis, I. Kontopoulos, A. Artikis, E. Alevizos, P. Delaunay, J.-E. Pouessel, R. Dreo, C. Ray, E. Camossi, A.-L. Jousselme, M. Hadzagic, Composite event patterns for maritime monitoring, in: Proceedings of the 10th Hellenic Conference on Artificial Intelligence (SETN), 2018.

[15] A. Margara, G. Cugola, G. Tamburrelli, Learning from the past: automated rule generation for complex event processing, in: Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, ACM, 2014, pp. 47–58.

[16] O.-J. Lee, J. E. Jung, Sequence clustering-based automated rule generation for adaptive complex event processing, Future Generation Computer Systems 66 (2017) 100–109.

[17] R. Mousheimish, Y. Taher, K. Zeitouni, Automatic learning of predictive cep rules: bridging the gap between data mining and complex event processing, in: Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, ACM, 2017, pp. 158–169.

[18] N. Katzouris, Scalable relational learning for event recognition, PhD Thesis, University of Athens.
URL `https://www.iit.demokritos.gr/sites/default/files/nkatz-phd-final.pdf`

[19] A. Dries, L. De Raedt, Towards clausal discovery for stream mining, in: ILP-2009, Springer, 2009, pp. 9–16.

[20] A. Srinivasan, M. Bain, An empirical study of on-line models for relational data streams, Machine Learning 106 (2) (2017) 243–276.

[21] N. Littlestone, Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm, Machine learning 2 (4) (1988) 285–318.

[22] T. N. Huynh, R. J. Mooney, Online structure learning for markov logic networks, in: ECML-2011, Springer, 2011, pp. 81–96.

[23] M. Richardson, P. Domingos, Markov logic networks, Machine learning 62 (1-2) (2006) 107–136.

[24] E. Michelioudakis, A. Skarlatidis, G. Paliouras, A. Artikis, Osla: Online structure learning using background knowledge axiomatization, in: ECML, Springer, 2016, pp. 232–247.

[25] N. Katzouris, E. Michelioudakis, A. Artikis, G. Paliouras, Online learning of weighted relational rules for complex event recognition, in: ECML/PKDD, 2018.

[26] A. Silva, C. Antunes, Multi-relational pattern mining over data streams, Data Mining and Knowledge Discovery 29 (6) (2015) 1783–1814.

[27] A. Srinivasan, M. Bain, Relational models with streaming ILP, in: ILP, 2013.

[28] N. Katzouris, A. Artikis, G. Paliouras, Parallel online learning of event definitions, in: International Conference on Inductive Logic Programming, Springer, 2017, pp. 78–93.

[29] N. A. Fonseca, A. Srinivasan, F. Silva, R. Camacho, Parallel ilp for distributed-memory architectures, Machine learning 74 (3) (2009) 257–279.

[30] A. Srinivasan, T. A. Faruquie, S. Joshi, Data and task parallelism in ilp using mapreduce, Machine learning 86 (1) (2012) 141–168.

[31] D. B. Skillicorn, Y. Wang, Parallel and sequential algorithms for data mining using inductive logic, Knowledge and Information Systems 3 (4) (2001) 405–421.

[32] N. A. Fonseca, F. M. A. Silva, V. S. Costa, R. Camacho, A pipelined data-parallel algorithm for ILP, in: 2005 IEEE International Conference on Cluster Computing (CLUSTER 2005), September 26 - 30, 2005, Boston, Massachusetts, USA, 2005, pp. 1–10.

[33] H. Ohwada, F. Mizoguchi, Parallel execution for speeding up inductive logic programming systems, in: International Conference on Discovery Science, Springer, 1999, pp. 277–286.

[34] J. H. Graham, C. D. P. Jr., A. H. Kamal, Accelerating the drug design process through parallel inductive logic programming data mining, in: 2nd IEEE Computer Society Bioinformatics Conference, CSB 2003, Stanford, CA, USA, August 11-14, 2003, 2003, pp. 400–402.

[35] A. K. Fidjeland, W. Luk, S. H. Muggleton, Customisable multi-processor acceleration of inductive logic programming, Latest Advances in Inductive Logic Programming (2014) 123–141.

[36] H. Nishiyama, H. Ohwada, Yet another parallel hypothesis search for inverse entailment, in: ILP, 2015.

[37] C. Rodrigues, H. Soldano, G. Bourgne, C. Rouveirol, Multi agent learning of relational action models, in: Proceedings of the Twenty-first European Conference on Artificial Intelligence, IOS Press, 2014, pp. 1087–1088.

[38] C. Rodrigues, H. Soldano, G. Bourgne, C. Rouveirol, A consistency based approach of action model learning in a community of agents, in: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 1557–1558.

[39] A. T. Vu, G. D. F. Morales, J. Gama, A. Bifet, Distributed adaptive model rules for mining big data streams, in: Big Data (Big Data), 2014 IEEE International Conference on, IEEE, 2014, pp. 345–353.