NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Robustness evaluation of Online Inductive Logic Programming Methods against Noisy Maritime Data

Emmanouil G. Lykos

Supervisors:
Panagiotis Stamatopoulos, Assistant Professor, NKUA
Alexander Artikis, Researcher, NCSR «Demokritos»
Nikos Katzouris, Associate Researcher, NCSR «Demokritos»

ATHENS

AUGUST 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αξιολόγηση στιβαρότητας των online μεθόδων επαγωγικού λογικού προγραμματισμού με θορυβώδη ναυτικά δεδομένα

Εμμανουήλ Γ. Λύκος

Επιβλέποντες:
Παναγιώτης Σταματόπουλος, Επίκουρος Καθηγητής, ΕΚΠΑ
Αλέξανδρος Αρτίκης, Ερευνητής, ΕΚΕΦΕ «Δημόκριτος»
Νίκος Κατζούρης, Ερευνητής, Εξωτερικός Συνεργάτης, ΕΚΕΦΕ «Δημόκριτος»

ΑΘΗΝΑ

ΑΥΓΟΥΣΤΟΣ 2020

BSc THESIS

Robustness evaluation of Online Inductive Logic Programming Methods against Noisy Maritime Data

Emmanouil G. Lykos

S.N.: 1115201600096

SUPERVISORS:
Panagiotis Stamatopoulos, Assistant Professor, NKUA
Alexander Artikis, Researcher, NCSR «Demokritos»
Nikos Katzouris, Associate Researcher, NCSR «Demokritos»

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αξιολόγηση στιβαρότητας των online μεθόδων επαγωγικού λογικού προγραμματισμού με θορυβώδη ναυτικά δεδομένα

Εμμανουήλ Γ. Λύκος

Α.Μ.: 1115201600096

ΕΠΙΒΛΕΠΟΝΤΕΣ:
Παναγιώτης Σταματόπουλος, Επίκουρος Καθηγητής, ΕΚΠΑ
Αλέξανδρος Αρτίκης, Ερευνητής, ΕΚΕΦΕ «Δημόκριτος»
Νίκος Κατζούρης, Ερευνητής, Εξωτερικός Συνεργάτης, ΕΚΕΦΕ «Δημόκριτος»

ABSTRACT

Inductive Logic Programming systems were widely used in order to help the task of producing accurate definitions of events of interest in various fields like maritime monitoring. In the field of maritime monitoring it is probable that the received data are mostly noisy, this means that the data will not be an accurate representation of the real-world, thus it is of outmost importance to evaluate these systems against noisy data, because these will be the closest to the real-world data that will be probably encountered. In this thesis, initially, a realistic and probabilistic noise injection method is proposed. Then, we inject the noise into the given maritime data in order to evaluate how much it corrupts the data. Finally, these systems are executed with the noisy dataset as input, in order to evaluate their robustness. From the aforementioned experiments, it was shown that our proposed method actually works, but affects differently every event. Moreover, it was illustrated that the aforementioned systems are robust and accurate against noisy data.

SUBJECT AREA: Event Recognition

KEYWORDS: Event Recognition, Inductive Logic Programming, Maritime Monitoring, Machine Learning

ΠΕΡΙΛΗΨΗ

Τα συστήματα Επαγωγικού Λογικού Προγραμματισμού έχουν ευρέως χρησιμοποιηθεί για να παράξουν αποτελεσματικά, ακριβείς ορισμούς για γεγονότα που μας ενδιαφέρουν σε διάφορα πεδία, όπως αυτό της Ναυτικής Παρακολούθησης. Στο συγκεκριμένο πεδίο είναι πολύ πιθανό ότι τα δεδομένα θα είναι θορυβώδη, αυτό σημαίνει ότι τα δεδομένα δεν θα είναι μία ακριβής αναπαράσταση του πραγματικού κόσμου, άρα είναι τεράστιας σημασίας να αξιολογήσουμε αυτά τα συστήματα σε θορυβώδη δεδομένα, διότι αυτά είναι πιο κοντά στα δεδομένα του πραγματικού κόσμου όπου συνήθως θα αντιμετωπίσει. Στην παρούσα πτυχιακή εργασία, αρχικά, παραθέτουμε μία ρεαλιστική και πιθανοτική μέθοδο εισαγωγής θορύβου. Έπειτα, εισάγουμε τον θόρυβο στα υπάρχοντα ναυτικά δεδομένα που έχουμε για να αξιολογήσουμε κατά πόσο αλλοιώνονται. Τέλος, εκτελούμε αυτά τα συστήματα με τα θορυβώδη δεδομένα ως είσοδο για να αξιολογήσουμε το πώς τα πάει σε αυτά. Από τα προαναφερθέντα πειράματα φάνηκε ότι η προτεινόμενη μέθοδος δουλεύει, αλλά επηρεάζει διαφορετικά το κάθε γεγονός. Επιπλέον, παρατηρείται ότι τα προαναφερθέντα συστήματα είναι στιβαρά και ακριβή απέναντι στα θορυβώδη δεδομένα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αναγνώριση Γεγονότων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Αναγνώριση γεγονότων, Επαγωγικός Λογικός Προγραμματισμός, Ναυτική Παρακολούθηση, Μηχανική Μάθηση

ACKNOWLEDGEMENTS

Undertaking and completing an undergraduate thesis is a challenging and difficult journey for most students because, for the majority of them, it is their first contact with research, thus, they should learn how research is conducted in order to complete their thesis. But, I was privileged enough to have good supervisors both as researchers, but mostly as people. Hence, I want to heartily thank my supervisors, Dr P.Stamatopoulos, Dr. A.Artikis and Dr. N.Katzouris for showing me how to work on the field of research and helping me completing the present thesis. Moreover, I want to further thank Dr. N.Katzouris, who was always there for me, answering with patience and politeness my numerous e-mails about issues that I encountered through the course of the thesis, even when he had a lot of work to do. Finally, I want to thank my friends and family who were always interested in my progress in this thesis and encouraging me, even in the hardest times.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Event Recognition is the task of recognizing events of interest, from data in various settings such as videos from surveillance cameras or vessel trajectories. In order to complete this task, a logic programming formalism named Event Calculus [12] is used. Event Calculus exploits the power of logic programming by using first-order logic rules -based on Event Calculus Dialect- in order to recognize the events of interest based on ground truth events that belong in the Knowledge Base. However, in huge datasets, many different patterns of the same event can occur and maybe after some time the actual behavior patterns of an event might change, thus, the task to write them by hand is tedious and error-prone because, as we mentioned above, even if the hand-crafted rules are accurate, the event might change its behavior and the pattern will not be anymore accurate. For this reason, Inductive Logic Programming(ILP)[4] methods, have been used, that their task is to learn the different patterns of an event of interest, given the Low Level and High Level Events that are present in some time and the information about whether the event, that we want to learn, happens or not. The Inductive Logic Programming methods that we dealt with, are the OLED[10] and WoLED[11, 15] which are shown to produce accurate patterns for the events to be learned and scale well. These methods, also, have the power that they are Online methods, specifically, they need only a single-pass of the training dataset to produce the results. However, these methods had not been evaluated on the maritime dataset, let alone their robustness against the maritime dataset that contains noise. In the current thesis, our task is to present realistic ways to inject noise in the maritime dataset, in order to evaluate the robustness of these Online Inductive Programming methods in a realistic setting that can be encountered in the real-world. The inspiration for the noise injection method came from Section 8.2 from [16] where the author expresses the idea of uncertainty of recognized events. Therefore, the author proposes a noise injection method that attaches to every recognized CAVIAR activity a probability, and afterwards, deletes the event instances that their attached probability is less than a given threshold. Our contribution to this method was to convert it to the maritime setting, not only by attaching to every AIS signal a probability but also each AIS signal was attached to its own threshold which derives in a realistic way. Afterwards, we will illustrate how much the recognized intervals that were evicted from Run-Time Event Calcucus(RTEC)[2], which is an Event Calculus framework, had changed. This will be done in order to determine how effective was the injected noise. Finally, we will present the results of OLED and WoLED in order to evaluate the robustness of these learners. The structure of the rest of the thesis is the following. Chapters 2,3 and 4 outlines the theoretical foundations that are necessary to understand the current thesis. Chapter 5, illustrates our methodology about injecting realistic noise in the maritime dataset. Chapter 6, presents the RTEC results in terms of how High Level Intervals are changed. Chapter 7, indicates the learning results from which we can evaluate the robustness of the aforementioned learners. Chapter 8, presents the conclusions of our work and subjects of further research.

2. BACKGROUND

2.1 First-Order Logic

In the present thesis, we use a first-order language where atoms, literals(possibly negated atom), rules and logic programs are defined as in [7] and not denotes negation as failure. Rules, atoms, literals and programs are ground if they contain no variables. Rules are written as $\alpha \leftarrow \delta_1, \delta_2, ..., \delta_n$ where α is an atom and $\delta_1, \delta_2, ..., \delta_n$ is a conjunction of literals. An interpretation I is a set of true ground atoms. I satisfies a ground literal α(respective not α) if and only if $\alpha \in I$(respective $\alpha \notin I$) and it satisfies a ground rule if and only if it satisfies the head, or does not satisfy the body. I is a minimal(Herbrand) model of a logic program Π if and only if it satisfies every ground rule in Π and none of its strict subsets has this property. I is an answer set of Π if and only if it is a minimal model of the program that results from the ground instances of Π, after removing all rules with a negated literal not satisfied by I, and all negative literals from the remaining rules. A choice rule is an expression of the form $\{\alpha\} \leftarrow \delta_1, \delta_2, ..., \delta_n$ and, intuitevely, denotes that whenever the body $\delta_1, \delta_2, ..., \delta_n$ is satisfied by an answer set I of a program that includes the choice rule, instances of the head α are arbitrarily included in I (satisfied) as well. A weak constraint is an expression of the form :$\sim \delta_1, \delta_2, ..., \delta_n.[w]$, where δ_i's are literals and w is an integer. The intuitive meaning of a weak constraint c is that the satisfaction of the conjunction $\delta_1, \delta_2, ..., \delta_n$ by an answer set I of a program that includes c incurs a cost of w for I. Inclusion of weak constraints in a program result in an optimization that returns answer sets of minimum cost. Formal account of choice rules and weak constraints' semantics are included in [7].

2.2 Event Calculus

Event Calculus[12] is a logic programming framework for representation and reasoning about effects and their events. Event Calculus consist of the following three entinties:

- Time Points: They denote the time that a predicate is true and usually have an integer value.
- Fluents: They are properties that can have different values at different time points.
- Events: They are events that with their occurences in time that can affect the value of one or more fluents.

Event Calculus is comprised by a set of rules that define the event occurences, the effects of events and the value of fluents which is called *event description*. In Table 2.1 are shown the main predicates of the Event Calculus and their meaning. In Table 2.2 are shown the domain-independent axioms of Event Calculus, which are the built-in rules of Event Calculus. Note that, *not* represents *negation by failure*. The first two describe the law of inertia, which tell that if an event was initiated then it still persists until something happens, that it terminates it. The first rule declares that an event holds at time $T+1$ if it is initiated at time T and second rule declares that an event holds at time $T+1$ if it holds at time T and was not terminated at this time. The third rule declares that an event holds

Table 2.1: Main Predicates of Event Calculus

Predicate	Meaning
happensAt(E, T)	Event E is occurring at time T
happensFor(E, I)	I is the list of the maximal intervals during which event E takes place
holdsAt($F = V, T$)	The value of fluent F is V at time T.
holdsFor($F = V, I$)	I is the list of the maximal intervals for which $F = V$ holds continuously
initiatedAt($F = V, T$)	At time T a period of time for which $F = V$ is initiated
terminatedAt($F = V, T$)	At time T a period of time for which $F = V$ is terminated
relative_complement_all(I', L, I)	I is the list of maximal intervals produced by the relative complement of the list of maximal intervals I' with respect to every list of maximal intervals of list L
union_all(L, I)	I is the list of maximal intervals produced by the union of the lists of maximal intervals of list L
intersect_all(L, I)	I is the list of maximal intervals produced by the intersection of the lists of maximal intervals of list L

Table 2.2: Domain Independent axioms of Event Calculus

Axioms
holdsAt($F = V, T + 1$) $\leftarrow$ initiatedAt($F = V, T$).
holdsAt($F = V, T + 1$) $\leftarrow$ holdsAt($F = V, T$), not terminatedAt($F = V, T$).
holdsAt($F = V, T$) $\leftarrow$ holdsFor($F = V, I$), $(T_s, T_e) \in I$, $T_s \leq T < T_e$.

at a time T if T belongs in a maximal interval that *holdsFor/2* predicate returns. The third rule is used for Statically Determined Fluents which will be explained later.

In RTEC event description defines the event instances with the use of *happensAt/2* predicate, the different values of fluents over time with the use of *holdsAt/2* and *holdsFor/2* predicates and the effects of the events with the use of *initiatedAt/2* and *terminatedAt/2* predicates. The events that are used in RTEC are shown in Table 2.1. Fluents in RTEC (that are used to define domain-dependent axioms) fall in these two categories: *simple* and *statically determined*. Simple fluents are fluents that their value is defined with the use of *initiatedAt/2* and *terminatedAt/2* predicates and the truth value of a *holdsAt/2* predicate is determined by using the law of inertia, specifically the first two built-in domain independent axioms of Table 2.2 are used. Statically Determined fluents are fluents that their value is defined with the use of *holdsFor/2* predicate and the truth value of *holdsAt/2* predicate is determined by if the given time belongs to any of the maximal intervals that *holdsFor/2* predicate returns, thus the third built-in domain independent axiom of Table 2.2 is used.

OLED input consists of two parts, narrative and annotation. Narrative consists of events that were defined with the *happensAt/2* predicate and their notation is that they define the input of the dataset. Annotation consists of High Level Events(HLEs) of interest that are defined by *holdsAt/2* predicates and declare the ground truth atoms of the event of interest, that the generated theory should satisfy.

2.3 Inductive Logic Programming

Inductive Logic Programming(ILP)[4] is a combination of inductive machine learning and logic programming and its purpose is to induce a theory which contains first order rules from observations. The main components that are involved in ILP are the following:

- The ground facts which are a set of positive examples E^+ and a set of negative examples E^-.
- The resulting theory H, which is set of rules.
- Background knowledge base B, which ia a set of rules that we consider them true.
- Language Bias(mode declarations) M which is used to make the ILP learners more efficient because they declare the literals-and their format- that can be used in the head and the body of the rules that the learned theory H should contain.

Therefore, the purpose of ILP learners are to find a theory H that satisfies all positive examples E^+ and none of the negative ones of set E^-. In order to achieve that, ILP learners work with a divide-and-conquer manner. Firstly, they construct rules that satisfy some of the positive examples and afterwards they apply their algorithm to the rest of the examples until every positive example is satisfied.

OLED uses ILP in order to learn new rules. Given a background knowledge B and mode declarations M, OLED's purpose is to find with a single-pass over the training set, domain-specific *initiatedAt/2* and *terminatedAt/2* rules that fit to training data and have good performance on test set. In Tables 2.3 and 2.4 example lines of the background knowledge and mode declarations file are shown. In the mode declaration file the first three lines at most times are used to declare the event of interest by declaring the format of the head of the resulting rules. The $+$ symbol declares that the instance that will be put in there is input instance and after that declares on what class it belongs. The $\#$ symbol declares that areaType is a constant. The last two lines are the format of the literals that would be put in the body of the resulting rules. Specifically, in this case would be placed in the body of the resulting rules only proximity and withinArea rules. In the background knowledge file it is declared that the instance that would be put inside the first parameter of loitering atom belongs in vessel1 class and that one of the value that can take the constant areaType is anchorage. Thus constant fields works as enumeration that their different values are defined in the background knowledge file.

Table 2.3: Background Knowledge example lines

fluent(loitering(X)) :- vessel1(X).
areaType(anchorage).

Table 2.4: Mode Declarations example lines

modeh(initiatedAt(loitering(+vessel1),+time))
modeh(terminatedAt(loitering(+vessel1),+time))
examplePattern(holdsAt(loitering(+vessel1),+time))
modeb(happensAt(proximity(+vessel1, +vessel1),+time))
modeb(happensAt(withinArea(+vessel1,#areaType),+time))

OLED applies in an Learning from Interpretations(LfI)[3] setting where each training example is a set of ground atoms, an interpretation. Generally speaking, in OLED, each resulting clause started from a general clause with empty body and was specialized as the learning process continued by gradually adding literals to its body.

2.4 Maritime Dataset

Maritime Dataset is split into two datasets. The former contains the Critical Points that were extracted from Trajectory Synopsis of the vessel trajectory dataset that contains the raw AIS Signals. Trajectory Synopsis is a framework that takes as input the raw AIS Signals that was received by a receiver and compresses them by keeping the most important points-the Critical Points- that are linked with a defined Low Level Event. This framework not only achieves 95% compression of the original dataset, but also does not corrupt the dataset much. Each Critical Point is represented by *coord* and *velocity* predicates which are the coordination and orientation fluents, respectively. Also, every Critical Point is attached with at least one Low Level Event(LLE) of Table 2.5 and its corresponding attributes.

Table 2.5: Low Level Events at Synopsis

stop_start	stop_end
slow_motion_start	slow_motion_end
gap_start	gap_end
change_in_speed_start	change_in_speed_end
entersArea	leavesArea
change_in_heading	

For example assume these four lines of our dataset presented on the following table

Table 2.6: Critical Point Representation Example

Event	Timestamp	Timestamp	MMSI	Attributes		
coord	1443650401	1443650401	228854000	-4.347	48.118	
velocity	1443650401	1443650401	228854000	0.0	0.0	257.0
change_in_heading	1443650401	1443650401	228854000			
gap_end	1443650401	1443650401	228854000			

These lines indicate the following at time 1443650401, for vessel with Maritime Mobile Service Identity(MMSI) number 228854000. Firstly, from the first line it is illustrated that vessel's current longitude and latitude are -4.347 and 48.118, respectively. Secondly, from the second line there are illustrated the current vessel's velocity attributes which are the speed, course over ground and true heading, respectively. Finally, the last two lines of the example in Table 2.6 indicate that at time 1443650401, vessel with MMSI 228854000 participates in a change_in_heading and a gap_end event.

The latter dataset contains the High Level Event(HLE) intervals that were extracted by executing RTEC with input the Critical Events dataset. The HLEs are presented in Table 2.7.

Table 2.7: High Level Events

withinArea	gap	stopped	lowSpeed
changingSpeed	movingSpeed	underWay	highSpeedNC
anchoredOrMoored	loitering	pilotBoarding	sarMovement
sarSpeed	sar	sarSpeed	trawlingMovement
trawlSpeed	trawling	tuggingSpeed	tugging

The dataset that we worked upon it and did experiments with it, was retrieved by the following workflow. Firstly, we collected the raw AIS signals from the AIS receivers from Brest port which is located in Brest, France. Afterwards, in order to get the Critical Points dataset we run Trajectory Synopsis on it. Finally, we got the HLE intervals by forwarding the Critical Points dataset into RTEC.

3. OLED BACKGROUND

3.1 Hoeffding Bound

Hoeffding Bound[8] is a statistical tool that may be used as a probabilistic estimator of the generalization error of a model(true expected error on the entire input), given its empirical error (observed error on a training subset)[5], which is of paramount importance in online learning. Hoeffding Bound is defined as follows: 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 $\epsilon = \sqrt{\frac{ln(1/\delta)}{2n}}$.

OLED uses the Hoeffding Bound to evaluate candidate specializations of a rule on a subset of training interpretations and not on the whole training set. Given an evaluation function G with range in $[0, 1]$ and a clause to be specialized r, OLED is using the Hoeffding Bound as follows. Assume also that after n training instances,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. $\Delta\overline{G} = \overline{G}(r1) - \overline{G}(r2) > 0$. Then by the Hoeffding bound we have that for the true mean ofthe scores' difference $\Delta\hat{G}$ it holds $\Delta\hat{G} > \Delta\overline{G} - \epsilon$, 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 to select at this point, with probability $1 - \delta$. In order to decide which specialization to select, it thus suffices to accumulate observations from the input stream until $\Delta\overline{G} > \epsilon$.

3.2 OLED

Before, the clause evaluation function is defined we should firstly define when an atom is True Positive(TP), False Positive(FP) and False Negative(FN). Let B is a set of domain-independent axioms, r be a clause and I an interpretation. $narrative(I)$ and $annotation(I)$ are the narrative and annotation part of I, respectively. We denote by M_I^r an answer set of $B \cup narrative(I) \cup r$. Given an anotation atom α we say that:

- α is a true positive(TP) atom w.r.t. clause r iff $\alpha \in annotation(I) \cap M_I^r$.
- α is a false positive(FP) atom w.r.t. clause r iff $\alpha \in M_I^r$ but $\alpha \notin annotation(I)$.
- α is a false negative(FN) atom w.r.t. clause r, iff $\alpha \in annotation(I)$ but $\alpha \notin M_I^r$.

Now that we know how TPs, FPs and FNs are derived we can define the clause evaluation function G. Given the TP, FP and FN counts of clause r, TP_r, FP_r and FN_r respectively the clause evaluation function G with range in $[0, 1]$ is the following:

$$G(r) = \begin{cases} \frac{TP_r}{TP_r + FP_r} & \text{,if } r \text{ is an initiatedAt clause} \\ \\ \frac{TP_r}{TP_r + FN_r} & \text{,if } r \text{ is an terminatedAt clause} \end{cases}$$

Generally speaking, when OLED encounters a False Predicted atom, either expands the theory by generating new rules, either specializes an existing rule based on the following cases:

- FN case. At least one FN atom α encountered. This may happen because:
  - No *initiatedAt/2* clause becomes true, failing to initiate the complex event that corresponds to α, when it should. Thus, generating a new *initiatedAt/2* clause , eliminates the FN atom turning it into a TP.
  - One or more *terminatedAt/2* clauses are over general, terminating the complex event that corresponds to α early. Thus, specializing the over-general *terminatedAt/2* clauses, the FN atom will be turned into a TP atom.
- FP case. At least one FP atom α encountered. This may happen because:
  - No *terminatedAt/2* clause becomes true, failing to terminate the complex event that corresponds to α, when it should, so α persists by inertia. Thus, generating a new *terminatedAt/2* clause eliminates the FP atom.
  - One or more *initiatedAt/2* clauses are over-general, re-initiating a corresponding complex event when they should not. Thus, specializing the existing over-general *initiatedAt/2* clauses the FP atom will be eliminated.

Therefore, the evaluation function $G(r)$ was defined that way because both *initiatedAt/2* and *terminatedAt/2* clauses affect the TP count of a theory H, therefore TP counts per clause are taken into account for the evaluation of both types of clauses. Moreover, specializing existing clauses further improves the quality of H by eliminating FPs in the *initiatedAt/2* case and FNs in favor of in favor of TPs in the *terminatedAt/2* case. Therefore, FPs(resp. FNs) should also be taken into account when evaluating *initiatedAt/2*(resp. *terminatedAt/2*) clauses. On the other hand, the total FP(resp. FN) count of a theory H is not affected by its existing *terminatedAt/2*(resp. *initiatedAt/2*) clauses, but instead requires new clauses to be generated.

The OLED algorithm for every incoming interpretation does the following:

1. New clauses are created in order to satisfy the most of the atoms that derived from the current interpretation and are not satisfied by the existing rules. The new rules ,initially, have empty body and they are getting specialized in the future.
2. Existing clauses are getting specialized by adding literals to their body using the Hoeffding Bound to choose the best specialization among the candidate specializations, with the way that was mentioned before.
3. Because some rules cannot be improved and have bad performance, OLED prunes these rules from theory with the use of Hoeffding Bound if rule's G-Score is lower than the quality threshold with probability $1-\delta$.

4. WOLED BACKGROUND

4.1 Online Structure Learning

Online Structure Learning with Background Knowledge Axiomization(OSL$_\alpha$)[13] builds on the OSL[9] algorithm for online learning of Markov Logic Networks(MLN). An MLN is a set of weighted first-order logic rules. Along with a set of domain constants, it defines a ground Markov network containing one feature for each grounding of a rule in the MLN, with the corresponding weight. Learning an MLN consists of learning its structure(the rules in the MLN) and their weights. OSL$_\alpha$ works by constantly updating the rules in an MLN in the face of new interpretations that stream-in, by adding new rules and updating the weights of existing ones. To infer query atoms' truth values, given an interpretation, OSL$_\alpha$ uses Maximum Aposteriori(MAP) inference[9], which amounts to finding the truth assignment to the query atoms that maximizes the sum of the weights of theory's rules satisfied by the interpretation. Specifically, given an interpretation I and a logic program Π that contains rules associated with real-valued weights, firstly, we find the maximal subset R_I of the weighted rules in Π that are satisfied by I. Then is assigned to the interpretation I a weight $W_\Pi(I)$ which is analogous to the sum of the weights of the rules in R_I, if I is an answer set of Π, else zero weight is assigned. Afterwards, these weights are normalized, in order to create a probability distribution over different answer sets. More formally, given that w_r denotes the weight of rule r and $ans(\Pi)$ is the set that contains the answer sets of R_I, $W_\Pi(I)$ is defined as

$$W_\Pi(I) = \begin{cases} \exp\left(\sum\limits_{r \in R_I} w_r\right), & \text{if } I \in ans(\Pi) \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

$$P_\Pi(I) = \frac{W_\Pi(I)}{\sum\limits_{J \in ans(\Pi)} W_\Pi(J)} \tag{4.2}$$

.Therefore, in MAP inference computes the most probable answer set of logic program Π which from equations (4.1) and (4.2), corresponds to finding an answer set $\mathcal{A}$ of Π such that

$$\mathcal{A} = \underset{I \in ans(\Pi)}{arg} \quad max P_\Pi(I) = \underset{I \in ans(\Pi)}{arg} \quad max W_\Pi(I) = \underset{I \in ans(\Pi)}{arg} \quad max \sum_{r \in R_I} w_r \tag{4.3}$$

But, this technique has the drawback that it only generates new rules, thus, many useless rules are present in the theory. Therefore, OSL$_\alpha$ relies on L_1-regularized weight learning, which in the long run, pushes the weights of non-useful rules to zero. Weight learning is also OSL$_\alpha$'s way to handle FP query atoms in the inferred state, which are due to erroneously satisfied rules in the current theory. OSL$_\alpha$ penalizes existing rules by using AdaGrad-based[6] weight learning technique, which supports L_1-regularization.

OSL$_\alpha$ might use specialize techniques for pruning the search space of the hypergraph structure to enhance efficiency, but the rule generation technique remains a bottleneck because searching for paths in the hypergraph is an expensive operation and blindly generating large sets of rules, which result in the increased cost of MAP inference during learning.

4.2 WoLED

WoLED is an OLED's extension and aims to improve the efficiency of online learning with the Event Calculus in MLN. Also, WoLED combines effectively OLED's rule learning with OSL$_\alpha$'s weight learning technique to jointly optimize together the rule's structure and its weight, contrary to OSL$_\alpha$ which a rule's structure is fixed and improvement in quality can be done only by changing its weight. To learn weights, WoLED replaces OLED's crisp logical inference with MAP inference and uses OSL$_\alpha$'s mistake-driven weight learning technique, which updates a rule's weight based on the query atoms that the rule misclassifies in the MAP-inferred state.

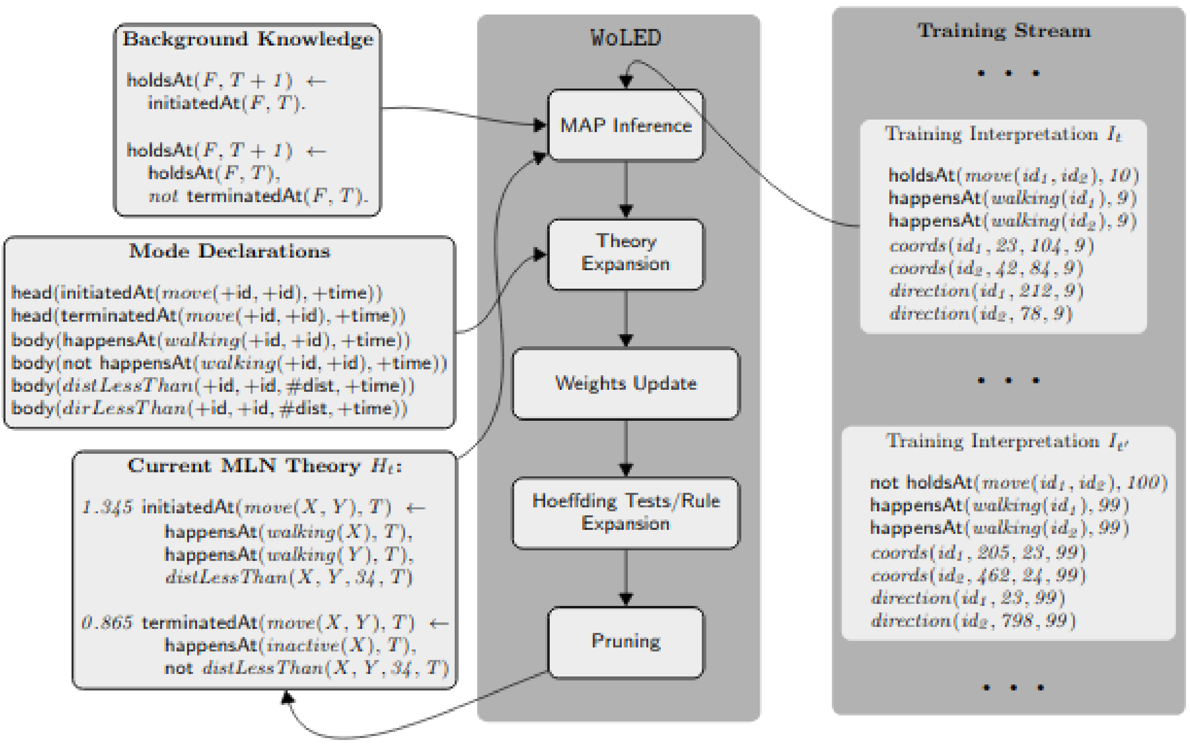

Background Knowledge
holdsAt(F, T + 1) ← initiatedAt(F, T).
holdsAt(F, T + 1) ← holdsAt(F, T), not terminatedAt(F, T).
Mode Declarations
head(initiatedAt(move(+id, +id), +time))
head(terminatedAt(move(+id, +id), +time))
body(happensAt(walking(+id, +id), +time))
body(not happensAt(walking(+id, +id), +time))
body(distLessThan(+id, +id, #dist, +time))
body(dirLessThan(+id, +id, #dist, +time))
Current MLN Theory H_t:
1.345 initiatedAt(move(X, Y), T) ← happensAt(walking(X), T), happensAt(walking(Y), T), distLessThan(X, Y, 34, T)
0.865 terminatedAt(move(X, Y), T) ← happensAt(inactive(X), T), not distLessThan(X, Y, 34, T)
WoLED
MAP Inference
Theory Expansion
Weights Update
Hoeffding Tests/Rule Expansion
Pruning
Training Stream
. . .
Training Interpretation I_t
holdsAt(move(id1, id2), 10)
happensAt(walking(id1), 9)
happensAt(walking(id2), 9)
coords(id1, 23, 104, 9)
coords(id2, 42, 84, 9)
direction(id1, 212, 9)
direction(id2, 78, 9)
. . .
Training Interpretation I_t'
not holdsAt(move(id1, id2), 100)
happensAt(walking(id1), 99)
happensAt(walking(id2), 99)
coords(id1, 205, 23, 99)
coords(id2, 462, 24, 99)
direction(id1, 23, 99)
direction(id2, 798, 99)
. . .

Figure 4.1: ***WoLED's High Level Strategy***

WoLED's high level strategy is illustrated in Figure 4.1. Specifically, for each incoming interpretation WoLED does the following:

1. Given the current interpretation and the background knowledge WoLED does MAP inference in order to get the inferred interpretation atoms. The MAP inference is done to "mature" rules, rules that were evaluated on a minimum number of examples to avoid change in weight to new rules that have not been tested thoroughly.

2. If a FN atom exists, the no rule in the current theory entails it, thus we should proceed to the Theory Expansion step to add a new rule r in the theory. This is done by using the FN atom α as a seed to generate the new rule r with combination to the given language bias.

3. Afterwards, the weight of each rule is updated based on their mistakes that were generated from the MAP inference that was previously done using the AdaGrad weight-updating technique.

4. The theory's rules that passed the Hoeffding test are getting specialized the same way that is done in OLED.

5. Low-quality rules that remained unchanged for a long period of time, set to the average number of $\mathcal{O}(\frac{1}{\epsilon^2} ln \frac{1}{\delta_h})$ examples for which the Hoeffding test has succeeded so far, and there is enough confidence, via an additional Hoeffding test, that its mean G-score is lower than a minimum acceptable G-score, are getting pruned from the theory.

4.3 WoLED ASP

The simple approach of WoLED that uses MLNs has the problem that the non-monotonic semantics of Event Calculus are incompatible with the open-world semantics of MLNs. Thus, the inference operation on Event Calculus-based MLN theories invokes superfluous operations, such as computing the completion of a theory[14], in order to endow the first-order logic representations on which MLNs rely with a non-monotonic semantics. This problem is addressed by WoLED's Answer Set Programming(ASP)[15] version by transforming WoLED's probabilistic inference with MLNs into an ASP optimization task -which naturally supports non-monotonic and commonsense reasoning- by using ASP solvers. Furthermore, this version of WoLED has the advantage that its methodology incorporates machine learning by learning the structure and the weights of the rules by using ASP tools. WoLED's ASP version has the same functionality as the MLN one, but they differ on the way that they generate the inferred state and learning new Complex Event patterns.

To generate the inferred state, WoLED uses MAP probabilistic inference which amounts to determine the most probable answer set $\mathcal{A}$ of a logic program $\Pi = B \cup H_t \cup I_t$, where B is static background knowledge, H_t is the current theory that WoLED learned at time t, and I_t is the interpretation(e.g. the input data) that WoLED received at time t. Therefore, we need to determine an answer set $\mathcal{A}$ of Π that satisfies equation (4.3), respectively, to find an answer set that maximizes the sum of weights of the satisfied rules, which is a classic weighted MaxSat problem that could be solved by traditional ASP solvers. However, we should transform the rules of current theory H_t, in order to be ready to handled by ASP solvers and produce optimal results. Firstly, because ASP solvers can optimize integer valued functions, the weights of the rules are converted to integers by firstly multiplying by some factor and afterwards they get normalized. Secondly, we produce a new theory $T(H_t)$ which is derived by transforming each H_t's rule in a form that is illustrated in[15] and gives to the solver the choice to satisfy some rules and not all of them. Finally, the optimal answer set $\mathcal{A}$ is determined from logic program $\Pi = B \cup T(H_t) \cup I_t$.

To expand the current theory H_t by adding new rules to it, this version of WoLED proceeds to the following steps. Firstly, a set of bottom rules is generated, using the constants in the erroneously predicted atoms to generate ground *initiatedAt/2* and *terminatedAt/2* atoms, which are placed in the head of initially empty-bodied rules. The bodies of these rules are afterwards populated by literals, grounded with constants that appear in the head, that are true in current data interpretation I_t, and are constraint to the language bias. Afterwards, constants in bottom rules are replaced with variables and are getting compressed, so the set of bottom rules $H_\perp$ will not have duplicates. Subsequently, the current theory H_t is transformed the same way it was on MAP inference, the set of bottom rules $H_\perp$ is transformed with a way illustrated in[15] and the new sets $T(H_t)$ and $T(H_\perp)$ are generated, respectively. Eventually, using ASP solvers, we determine the best answer set for logic program Π, which contains some optimization rules and $B \cup I_t \cup T(H_t) \cup T(H_\perp)$. Finally, from $H_\perp$, the rules and literals that are not used to the returned answer set are deleted from $H_\perp$ and H_{new} is the new $H_\perp$, while the current theory equals to $H_t \cup H_{new}$. After, the generation of new patterns, their weights are updated based on their groundings in I_t and the true state. Moreover, each new rule r is associated with the bottom rules from $H_\perp$, which are θ-subsumed by r_i. These bottom rules are used as a pool of literals for further specializing r over time.

Note, that in the experiments that will be presented afterwards this version of WoLED is used.

5. NOISE INJECTION

5.1 Noise Injection Method

The authors of [16] at section 8.2, in order to inject noise to the dataset, propose the idea that every received Critical Point is not getting retrieved with complete certainty, thus the points that are received with a certainty lower than a certain standard should be getting removed from the dataset. Although we incorporated that idea in our noise injection method, this exact method has two drawbacks. Firstly, this method was intended to be applied in the CAVIAR dataset which probably have different form than the Maritime dataset that we currently work with. Secondly, this method defines a global certainty standard, such as 70% and just for each point, takes a random number in $[0, 1]$ and if that number is less than 0.7, then the point gets deleted. But, by deleting Critical Points that way is not very realistic bacause every point has its own attributes on how it is derived, thus it is not necessary that all the points have the same retrieval uncertainty. Therefore, we need to define a realistic way to attach every Critical Point to its own level of uncertainty.

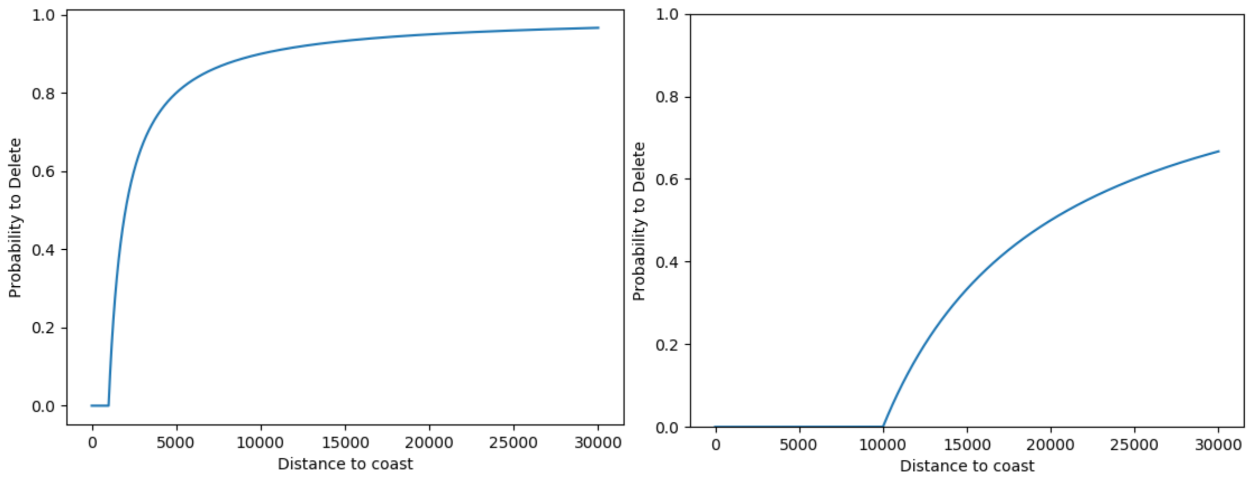
One factor that we thought that was decisive enough to inform about the level of uncertainty of a given Critical Point, was the distance of that point from the coastline because every AIS receiver has a specific distance range. Thus, when a specific receiver receives a signal from a great distance, like from the Atlantic Ocean, probably that signal might be corrupted because it might took much time to reach the receiver. Initially, we thought to define the uncertainty of a given signal by making distance levels. For example if a signal has distance in $[0, 1000]$ meters will be completely certain, if the signal has distance in $[1000, 5000]$ meters then will have certainty equal to 0.8, etc. However, this method does not provide continuity to the certainty value, and in order to evaluate our method's effectiveness we might need to tune every interval of each distance level, where in fact will be a painstaking process because there are millions of possible intervals and certainty values to attach to them. Instead, we thought to produce the certainty of every Critical Point in a dynamic and continuous way by defining an increasing and continuous function that takes as input the distance of the Critical Point from the coast and returns its uncertainty. This function will be analyzed afterwards.

In order to apply the aforementioned method, firstly, we attached to each *coord*[1] atom a probability, which denotes how likely is to delete its corresponding LLE atoms from the data, because the *coord* atom was attached with some LLEs. As we mentioned before, in order to completely define our method we have to declare an increasing function $f : (0, +\infty) \to [0, 1]$ that takes as argument the distance of the current point from the coast and returns the probability that this point's LLEs will be deleted. The function that we thought is the following:

$$f(x) = \begin{cases} 0, & \text{if } c \geq x \\ 1 - \frac{c}{x}, & \text{else} \end{cases}$$

where the parameter $c \in (0, +\infty)$ denotes the max distance that surely a transmitter's message is not noisy, so it will not get deleted.

[1]The *coord* atom contains information about the location of a vessel in some time.



(a) Noise Function with c $= 1000$ **(b) Noise Function with c $= 10000$**

Figure 5.1: Noise Function

We chose this function for the following reasons. Firstly, because we can declare a distance threshold, such as, when the distance is below that threshold, the AIS signal that was received was not noisy, which is realistic because every AIS receiver within some distance range receives completely clear signals. Secondly, as we can see in Figure 5.1a and 5.1b ,as the parameter c increases, the growth rate of the function decreases, which we claim that it is realistic because if one receiver can receive a clear signal within a greater range than some other, then the former receiver is more robust to corrupted signals that came from a distance greater than the distance threshold. Finally, contrary to our first suggestion about setting uncertainty, that we not only have to define the intervals but also their corresponding uncertainty values this method has only one parameter to tune, which is the distance threshold. Therefore, we can make easier and more experiments because it will be understandable on why we chose these values for our parameter and what that parameter actually means.

After we attached to each *coord* atom a probability, we take for each of them a random number from $[0, 1]$ with uniform distribution and if the generated number is below the attached probability we delete the *coord* atom and all the atoms that have the same MMSI and Timestamp because these atoms were attached with the deleted *coord* atom because they form a critical point[2] in the dataset.

5.2 Technical Notes

In order to be able to inject the noise we wrote a *Python3* script and we used the following libraries:

- Fiona, which is a Python package that helps programmers to read and write geospatial data from various file formats, in order to be integrated into programs and work harmonically with other packages that handle geospatial data, such as Shapely. In the present noise injection method, Fiona is used to read the shapefile that contained the European Coastline, in order to extract its corresponding Linestring.
- Shapely, which is a Python package that works as a wrapper for Java Topology

[2]**critical point** of a trajectory is a point where we recognized an LLE in it.

Suite(JTS), is used to create and manipulate planar geometrical objects, in order to answer effectively spatial queries. In the present noise injection method, Shapely is used to find the nearest point of the coastline or from a set of points, from the current location of the vessel's trajectory. This is achieved by saving the coastline and the set of points as LineString and MultiPoint objects, respectively.

- Geopy is a client for geocoding web services. Geopy is used by Python developers to locate the coordinates of addresses, cities, countries, and landmarks across the globe using third-party geocoders and other data sources. In the present method, Geopy is used to find the Great Circle Distance between the current location of the vessel's trajectory and its nearest coastline point.
- Sklearn, is a Python library that is used for machine learning because it contains implementations for all the popular machine learning algorithms such as Random Forest Regression and Classification, K-Means Clustering, Principal Component Analysis etc. Sklearn is used in the optimization step of the present method, because the Mini-Batch K-Means[1] is used.

Applying naively that method, we encountered the problem that it took too much time to be applied in the whole dataset. Thus, there was a need to make that method more efficient by applying an optimization. Visualizing the data on a map, we observed that there were many points close to each other, probably because the AIS signals were frequent and there were present many vessels on the sea. Thus, we probably made almost the same nearest points query many times because two points that are close to each other probably have almost the same nearest coastline point. Therefore, we developed an optimization to our noise injection method whose main concept is that points which are in some defined range from its nearest center point have the same nearest coastline point with this center point. In practice, this optimization helps us do the nearest points query between the current point and the coastline fewer times, which makes the execution of the method faster because the aforementioned query is very costly because the European coastline is large. Specifically, the optimized method to find the distances works as follows.

Firstly, runs in the whole dataset the Mini Batch K-Means algorithm(because in huge datasets this algorithm is efficient), in order to have K good initial centers, so the aforementioned query will be done fewer times than just incrementally adding the centers into the data structure. Afterwards, these initial centers are stored into a hash table with keys the coordinates of the centers and values their nearest coastline point.

Then, for each incoming point, we determine its nearest neighbor from the keys of the hash table(by asking a query between a point and a MultiPoint instance) and if their distance is less than the defined threshold the current point, we will assume that its nearest coastline point is the same as its nearest neighbor. Else, we do the nearest point query between the given point and the coastline and find their distance. If the hash table has fewer records than a user-defined limit then the point-nearest coastline point pair is inserted into the hash table. This is done because when the hash table is huge the optimization algorithm is useless.

Finally, we attach to the coord atom a probability based on the distance found, then, we generate a random number, and if it is less than the given probability the coord atom and its respective LLEs are deleted from the dataset. Last but not least, it is worth mentioning that the presented optimization algorithm will have a trade-off between precision and speed, but, as we mentioned above, if it is not applied, the noise injection script will take even days to finish because the maritime dataset is huge. A pseudocode of the mentioned method is illustrated below, but note, that it is assumed that each critical point contains all the LLEs that are happening at it, so we do not need to check whether it is a *coord* atom.

Algorithm 1 Noise Injection Method

```
Input: Set of critical points $P$, Coastline as MultiLineString $CL$, Maximum number of Hash Table keys $k_{max}$, number of initial centers $k_{init}$, Proximity threshold $d_{max}$, AIS Receiver range $d_{range}$
Returns: A new set of critical points after noise injection $P_{new}$
$CurrHashTableRecords = 0$
$P_{new} = \emptyset$
$PointCoastlineNearestMap = HashMap < Point, Point > ()$
$C_{init} = MiniBatchKMeans(k_{init}, P)$

for all $c \in C_{init}$ do
  $NearestCLPoint = NearestPoint(c, CL)$
  $PointCoastlineNearestMap[c] = NearestCLPoint$
  $CurrHashTableRecords = CurrHashTableRecords + 1$
end for

for all $p \in P$ do
  $NearestCenterPoint = NearestPoint(p, PointCoastlineNearestMap.keys())$
  $CurrentDistance = Distance(p, NearestCenterPoint)$

  if $CurrentDistance \leq d_{max}$ then
    $PointDistance = Distance(p, PointCoastlineNearestMap[NearestCenterPoint])$
  else
    $NearestCLPoint = NearestPoint(p, CL)$
    $PointDistance = Distance(p, NearestCLPoint)$

    if $CurrHashTableRecords < k_{max}$ then
      $PointCoastlineNearestMap[p] = NearestCLPoint$
      $CurrHashTableRecords = CurrHashTableRecords + 1$
    end if

    if $PointDistance < d_{range}$ then
      $P_{new} = P_{new} \cup \{p\}$
    else
      $DeleteProbability = 1 - d_{range}/_{PointDistance}$

      if $Random() \geq DeleteProbability$ then
        $P_{new} = P_{new} \cup \{p\}$
      end if
    end if
  end if
end for

return $P_{new}$
```

6. RTEC EVALUATION

In order to evaluate OLED's and WOLED's robustness against noisy data, firstly, we should inject noise to the Critical Points Dataset -which is described in Section 2.4-, using the method described in the previous chapter. Afterwards, we should run RTEC with the aforementioned dataset and the HLE patterns as input to recognize the new HLE intervals that were derived from the noisy dataset. Finally, we pass to the Inductive Programming learners the clean Critical Points dataset and the HLE intervals dataset -which is described in Section 2.4-, in order to evaluate their robustness, by applying in an online manner our noise injection method.

But, before we execute our experiments in OLED and WOLED it is useful to compare the clean HLE intervals-that are used as ground truth intervals- with the noisy ones and observe how different they are, for three reasons. Firstly, because we should check if our noise injection method actually works by producing reasonable results,thus, by observing how the values of evaluation metrics are changing by getting through different values of the distance threshold. Secondly, we can observe how much the aforementioned noise injection method affects the recognized intervals of different types of HLEs, such as HLEs with more complex patterns or, HLEs that are defined as Statically Determined or Simple Fluents. Finally, the experiments that will be performed on OLED and WOLED, will evaluate their robustness against noisy data. This attribute will be tested by observing how good are the patterns learned by some HLEs of interest on different variations of noisy dataset. But, in order to check if the learners are really robust, we ideally want to choose HLEs that their recognized intervals were actually changed much by noise injection, because if not, then it will be like to executing the learner twice with the clean dataset, which is something we do not want, because we will think that the learner is robust, but actually the noise that will be applied to the clean dataset will not be detrimental at learning the pattern of the event of interest.

Before we show the results of our work, note that each HLE interval's time unit(in our approach, time unit is equal to a millisecond) that is in the noisy or clean HLE intervals dataset can be classified as **true positive**, **false positive** and **false negative** by the following rules:

- **true positive** is an interval's millisecond that belongs to the clean dataset interval and in some of its corresponding [1] noisy dataset intervals.
- **false positive** is an interval's millisecond that belongs to the noisy dataset interval but does not belong in some of its corresponding clean dataset intervals.
- **false negative** is an interval's millisecond that belongs to the clean dataset interval but does not belong in some of its corresponding noisy dataset intervals.

The results presented have different values of the noise function's parameter c and we found 350 initial centroids while the Hash Table can hold 500 entries and d_{near} equals ten thousand meters.

The approximate results for different values of c that we got, running this experiment, are the following:

[1]**corresponding intervals** are two intervals that represent the same fluent, the same MMSI, the same argument(if exists) and the same value.

Table 6.1: *anchoredOrMoored*

c	TPs	FPs	FNs	Precision	Recall	F1-Score
1000	25.46 10^7	66.61 10^6	43.81 10^6	0.7920	0.8527	0.8212
2500	28.59 10^7	14.37 10^6	11.53 10^6	0.9521	0.9612	0.9567
5000	29.56 10^7	99.57 10^5	18.18 10^5	0.9674	0.9939	0.9805
7500	29.74 10^7	10.91 10^6	0	0.9646	1.0	0.9820
10000	29.74 10^7	69.35 10^5	0	0.9772	1.0	0.9885
12500	29.74 10^7	18.51 10^3	0	0.9999	1.0	0.9999
15000	29.74 10^7	54.61 10^3	0	0.9998	1.0	0.9999
17500	29.74 10^7	0	0	1.0	1.0	1.0
20000	29.74 10^7	0	0	1.0	1.0	1.0

Table 6.2: *gap*

c	TPs	FPs	FNs	Precision	Recall	F1-Score
1000	67.95 10^8	25.14 10^8	22.2 10^9	0.7299	0.2344	0.3548
2500	99.29 10^8	28.24 10^8	19.07 10^9	0.7785	0.3424	0.4756
5000	13.29 10^9	34.62 10^8	15.71 10^9	0.7933	0.4582	0.5809
7500	14.96 10^9	32.49 10^8	14.03 10^9	0.8216	0.5161	0.6340
10000	16.64 10^9	36.32 10^8	12.36 10^9	0.8208	0.5738	0.6754
12500	17.92 10^9	35.54 10^8	11.08 10^9	0.8345	0.6179	0.7100
15000	18.65 10^9	36.91 10^8	10.35 10^9	0.8348	0.6432	0.7266
17500	19.4 10^9	31.86 10^8	95.95 10^8	0.8590	0.6691	0.7522
20000	19.99 10^9	34.25 10^8	90.07 10^8	0.8537	0.6894	0.7628

Table 6.3: *loitering*

c	TPs	FPs	FNs	Precision	Recall	F1-Score
1000	26.88 10^7	17.37 10^8	36.9 10^7	0.1340	0.4214	0.2034
2500	34.2 10^7	12.13 10^8	29.58 10^7	0.2199	0.5363	0.3119
5000	39.2 10^7	10.85 10^8	24.57 10^7	0.2654	0.6147	0.3707
7500	41.89 10^7	89.54 10^7	21.89 10^7	0.3187	0.6568	0.4292
10000	44.58 10^7	71.03 10^7	19.19 10^7	0.3856	0.6990	0.4970
12500	51.28 10^7	83.14 10^7	12.5 10^7	0.3815	0.8040	0.5175
15000	52.89 10^7	72.87 10^7	10.88 10^7	0.4206	0.8294	0.5581
17500	42.64 10^7	63.57 10^7	21.14 10^7	0.4015	0.6685	0.5017
20000	52.92 10^7	55.69 10^7	10.86 10^7	0.4872	0.8297	0.6139

Table 6.4: *lowSpeed*

c	TPs	FPs	FNs	Precision	Recall	F1-Score
1000	12.2 10^7	14.15 10^8	26.72 10^7	0.0794	0.3135	0.1267
2500	18.56 10^7	10.27 10^8	20.36 10^7	0.1530	0.4769	0.2317
5000	20.48 10^7	91.12 10^7	18.44 10^7	0.1835	0.5263	0.2722
7500	23.75 10^7	78.76 10^7	15.17 10^7	0.2317	0.6102	0.3358
10000	24.58 10^7	62.72 10^7	14.34 10^7	0.2816	0.6317	0.3895
12500	28.64 10^7	68.21 10^7	10.28 10^7	0.2957	0.7358	0.4219
15000	29.99 10^7	64.92 10^7	89.34 10^6	0.3159	0.7705	0.4481
17500	23.48 10^7	53.22 10^7	15.44 10^7	0.3061	0.6032	0.4061
20000	30.63 10^7	47.64 10^7	82.91 10^6	0.3913	0.7870	0.5227

Table 6.5: *pilotOps*

c	TPs	FPs	FNs	Precision	Recall	F1-Score
1000	22.83 10^3	14.25 10^3	11.57 10^3	0.6157	0.6637	0.6388
2500	33.48 10^3	81.3 10^1	91.4 10^1	0.9763	0.9734	0.9749
5000	34.4 10^3	34	0	0.9990	1.0	0.9995
7500	34.4 10^3	31	0	0.9991	1.0	0.9995
10000	34.4 10^3	38	0	0.9989	1.0	0.9994
12500	34.4 10^3	28	0	0.9992	1.0	0.9996
15000	34.4 10^3	0	0	1.0	1.0	1.0
17500	34.4 10^3	0	0	1.0	1.0	1.0
20000	34.4 10^3	28	0	0.9992	1.0	0.9996

Table 6.6: *rendezVous*

c	TPs	FPs	FNs	Precision	Recall	F1-Score
1000	69.29 10^5	17.62 10^4	83.32 10^3	0.9752	0.9881	0.9816
2500	70.02 10^5	15.26 10^3	10.99 10^3	0.9978	0.9984	0.9981
5000	70.11 10^5	54.84 10^2	12.94 10^2	0.9992	0.9998	0.9995
7500	70.12 10^5	58.04 10^2	10.81 10^2	0.9992	0.9998	0.9995
10000	70.12 10^5	4	10.81 10^2	0.9999	0.9998	0.9999
12500	70.12 10^5	0	10.81 10^2	1.0	0.9998	0.9999
15000	70.12 10^5	0	10.81 10^2	1.0	0.9998	0.9999
17500	70.12 10^5	11	54.4 10^1	0.9999	0.9999	0.9999
20000	70.13 10^5	40	7	0.9999	0.9999	0.9999

Table 6.7: *stopped*

c	TPs	FPs	FNs	Precision	Recall	F1-Score
1000	63.41 10^7	61.22 10^7	25.33 10^7	0.5088	0.7146	0.5944
2500	74.92 10^7	25.27 10^7	13.82 10^7	0.7478	0.8443	0.7931
5000	81.11 10^7	20.33 10^7	76.31 10^6	0.7996	0.9140	0.8530
7500	81.34 10^7	13.14 10^7	73.94 10^6	0.8609	0.9167	0.8879
10000	83.8 10^7	10.34 10^7	49.4 10^6	0.8902	0.9443	0.9164
12500	85.31 10^7	18.53 10^7	34.3 10^6	0.8216	0.9613	0.8860
15000	86.74 10^7	11.15 10^7	19.95 10^6	0.8861	0.9775	0.9296
17500	81.02 10^7	11.99 10^7	77.19 10^6	0.8711	0.9130	0.8916
20000	84.51 10^7	11.76 10^7	42.3 10^6	0.8778	0.9523	0.9136

Table 6.8: *withinArea*

c	TPs	FPs	FNs	Precision	Recall	F1-Score
1000	52.64 10^8	84.91 10^8	49.54 10^8	0.3827	0.5152	0.4392
2500	59.45 10^8	72.43 10^8	42.73 10^8	0.4508	0.5818	0.5080
5000	63.4 10^8	62.07 10^8	38.78 10^8	0.5053	0.6205	0.5570
7500	67.02 10^8	58.24 10^8	35.16 10^8	0.5350	0.6559	0.5893
10000	70.45 10^8	50.77 10^8	31.73 10^8	0.5812	0.6895	0.6307
12500	72.99 10^8	45.77 10^8	29.19 10^8	0.6146	0.7143	0.6607
15000	73.33 10^8	43.35 10^8	28.85 10^8	0.6285	0.7176	0.6701
17500	78.26 10^8	42.55 10^8	23.92 10^8	0.6478	0.7659	0.7019
20000	79.04 10^8	36.93 10^8	23.14 10^8	0.6816	0.7735	0.7246

Table 6.9: Total results

c	TPs	FPs	FNs	Precision	Recall	F1-Score
1000	31.93 10^9	35.68 10^9	70.76 10^9	0.4722	0.3109	0.3750
2500	41.08 10^9	33.54 10^9	61.6 10^9	0.5505	0.4001	0.4634
5000	48.75 10^9	32.96 10^9	53.93 10^9	0.5966	0.4747	0.5287
7500	54.43 10^9	31.83 10^9	48.25 10^9	0.6310	0.5301	0.5761
10000	58.43 10^9	30.97 10^9	44.25 10^9	0.6536	0.5690	0.6084
12500	62.88 10^9	29.3 10^9	39.8 10^9	0.6821	0.6124	0.6454
15000	65.29 10^9	28.07 10^9	37.4 10^9	0.6993	0.6358	0.6661
17500	68.18 10^9	26.79 10^9	34.5 10^9	0.7180	0.6640	0.6899
20000	70.0 10^9	25.38 10^9	32.69 10^9	0.7339	0.6817	0.7068

Therefore, we can easily observe that as c decreases, then the F_1-Score also decreases. This is obvious because when we delete Low Level Event atoms then the recognized intervals might be different, thus there would be error, and as the number of atoms deleted increases, the error will get increased. Hence, we can conclude that the noise injection method indeed works because the results are reasonable.

Observing, the F_1-Scores of each event individually we observe that the noise does not affect negatively event's F_1-Score to the same degree. For example the rendezVous event keeps an almost perfect F_1-Score even for $c = 1000$, but pilotOps event for $c = 20000$ keeps an almost perfect F_1-Score, while for $c = 1000$ has a low F_1-Score. We theorize that this happens because each High Level Event(HLE) definition has not the same complexity, therefore have not the same robustness to noise. Furthermore, we observe that Simple Fluents are less robust than Statically Determined Fluents. For instance, the *lowSpeed* fluent which has the following definition,

$$\begin{aligned}
&initiatedAt(lowSpeed(Vessel) = true, T) \leftarrow \\
&\qquad happensAt(slow\_motion\_start(Vessel), T).\\
&terminatedAt(lowSpeed(Vessel) = true, T) \leftarrow \\
&\qquad happensAt(slow\_motion\_end(Vessel), T).\\
&terminatedAt(lowSpeed(Vessel) = true, T) \leftarrow \\
&\qquad happensAt(start(gap(Vessel) = \_Status), T).
\end{aligned}$$

,is a Simple Determined Fluent and we could think that this HLE is not very robust-because it depends on the truth value of an LLE which might be deleted-, which indeed is not, because we can see in the corresponding table that its F_1-score begins from 0.52 and falls to 0.12, so the F_1-Score begins with a low value and as we decreasing c it becomes even lower. On the contrary, the *rendezVous* fluent, which has the following definition

$$\begin{aligned}
&holdsFor(rendezVous(Vessel_1, Vessel_2) = true, I) \leftarrow \\
&\quad holdsFor(proximity(Vessel_1, Vessel_2) = true, I_p),\\
&\quad not\ oneIsTug(Vessel_1, Vessel_2),\\
&\quad not\ oneIsPilot(Vessel1, Vessel_2),\\
&\quad holdsFor(lowSpeed(Vessel_1) = true, I_{l_1}),\\
&\quad holdsFor(lowSpeed(Vessel_2) = true, I_{l_2}),\\
&\quad holdsFor(stopped(Vessel_1) = farFromPorts, I_{s_1}),\\
&\quad holdsFor(stopped(Vessel_2) = farFromPorts, I_{s_2}),\\
&\quad union\_all([I_{l_1}, I_{s_1}], I_{1_b}),\\
&\quad union\_all([I_{l_2}, I_{s_2}], I_{2_b}),\\
&\quad intersect\_all([I_{1_b}, I_{2_b}, I_p], I_f), I_f \neq [],\\
&\quad holdsFor(withinArea(Vessel_1, nearPorts) = true, I_{w_1}),\\
&\quad holdsFor(withinArea(Vessel_2, nearPorts) = true, I_{w_2}),\\
&\quad holdsFor(withinArea(Vessel_1, nearCoast) = true, I_{w_3}),\\
&\quad holdsFor(withinArea(Vessel_2, nearCoast) = true, I_{w_4}),\\
&\quad relative\_complement\_all(I_f, [I_{w_1}, I_{w_2}, I_{w_3}, I_{w_4}], I).
\end{aligned}$$

,is a Statically Determined Fluent and we could think that this HLE is very robust because it is pretty complex, which indeed is, because we can see that even with the ultimate form of noise the F_1-Score is 0.98 which is almost perfect. Furthermore, this might happen because the definition of Simple Fluents-by *initiatedAt/2* and *terminatedAt/2* predicates-contain *happensAt/2* or *holdsAt/2* predicates that check the presence of an LLE atom that appears on a critical point-at a particular time point- which can be directly deleted by the aforementioned noise injection method, while the definition of Statically Determined Fluents-by *holdsFor/2* predicates-contain *holdsFor/2* predicates that check the value of a HLE Fluent in a particular time interval, and interval handling predicates, thus it is more difficult to change these intervals because the underlying LLEs that determine the truth value of that HLE should be deleted.

7. LEARNING EXPERIMENTS

7.1 Data Preparation

In order to be able to run the experiments, we had to convert our data to OLED's and WOLED's desired format. Furthermore, instead of having to run our Python3 script in order to inject the noise and waiting extra time to run our experiments, we incorporated our noise injection method into our program that reads the datasets and converts it to the desired format. Also, note that, that in this program there was no need to apply the optimization of our noise injection method, because it was seen that it was not detrimental to learners' performance, probably because it was applied in an online manner. Moreover, before the learners start, they read the HLE intervals dataset and save them into an interval tree, which is a data structure that handles intervals.

Therefore, the algorithm that retrieved each interpretation(or batch) of size n is the following:

- Reads the Critical Points dataset until it had read n different timestamps, and for each coord atom that encounters, the noise injection method is applied in order to check if its corresponding LLEs will be deleted.
- The LLEs that will not get deleted, are converted into *happensAt/2* predicates and are added to the narrative.
- Assume that $batchLow$ and $batchHigh$ are the lowest and greatest timestamps of the current batch respectively. Thus, we make a query into the interval tree to get the HLE intervals that intersect the $[batchLow, batchHigh]$ interval.
- Then for each HLE interval, we convert it into *happensAt/2* or *holdsAt/2* predicate if it should be on narrative or annotation-in which they will be added-, respectively, unless they should be on narrative and should be deleted. This is done for all the batch's timestamps that are in the current HLE interval.
- Finally, the final narrative and annotation lists are returned.

7.2 Performance Evaluation

As in every machine learning model, we should evaluate its performance on validation data that are used as the testing set. In this current experiment we use k-fold cross validation, in order to evaluate the performance of our model.

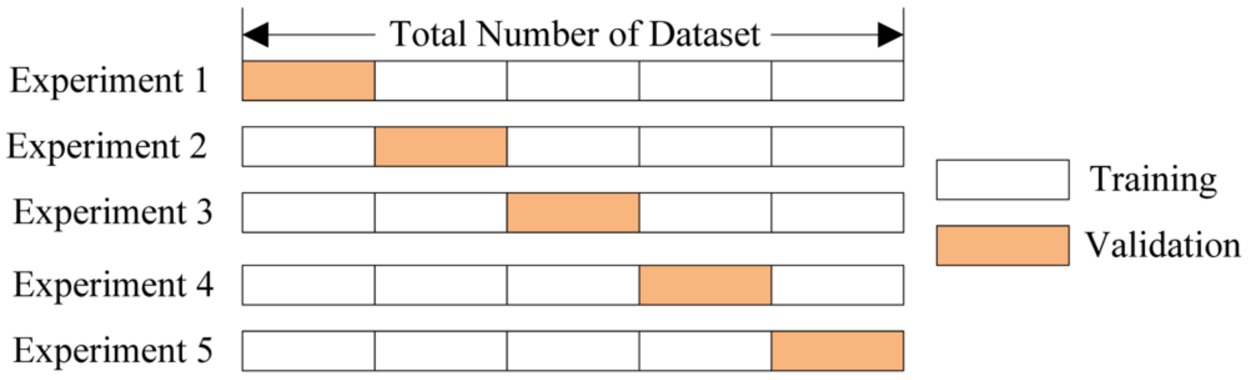


Figure 7.1: k-Fold Cross Validation

In order to split our dataset into k folds we follow these steps:

- Firstly, we get the n different timestamps that are present on our Critical Points dataset.
- Then, we split these discrete timestamps into k equal-sized parts and each part is represented by an interval which starts at its first timestamp(except the first fold which starts at 0) and ends at the first timestamp of its next fold(except the last fold which never ends)
- Finally, every HLE interval from the corresponding dataset, if it completely falls into a specific fold is kept into that fold, else it was split properly, so every part of it belongs to the right fold.

The aforementioned split method might show the problem that some fold might not have many annotation samples of an event that the learners tried to learn, but, in practice, it was shown that even the HLE intervals of the event of interest were evenly splitted, thus, we never encountered this problem.

In this learning experiment, we used the dataset's initial segment of 1,500,000 time points for training and testing.

The experimental setting was a 5-fold cross-validation process. At each fold, and for each complex event in the experiment, we formed a training set by sampling 30,000 data batches of size 50 from the entire dataset. A size-50 training batch consists of data corresponding to 50 consecutive time points sampled from some region of the dataset where the target complex event occurs (i.e. it is initiated, terminated or holds continuously). Note that although these training batches certainly contain positive examples for the target complex events (e.g. for its initiation/termination), they almost always contain negative examples as well, as they reference pairs of vessels for which the target complex event does not hold. To form a testing set for each fold of the cross-validation process we sampled 10,000 data batches of size 50 (with no overlap with the training batches) in a fashion similar to that of the training set. The F1-scores that we report are micro-averages from the 5-fold cross-validation process, meaning that TPs, FPs and FNs from the testing sets of all 5 runs where aggregated and an F1-score was calculated from these values.

At each fold, we compare OLED's and WOLED's performance in the noisy and the non-noisy case. Exactly the same training and testing sets were sampled in both cases (non-noisy, noisy), i.e. at each fold we have a noise-free training-testing pair and its noisy versions for different values of the c constant that controls the amount of noise.

At each run of each fold the training batches were treated as a stream, therefore both OLED and WOLED were allowed a single pass of these data.

7.3 Experimental Setting and Results

Now that we evaluated the influence of noise injection and defined our evaluation methods to the recognition of HLE intervals we are ready to evaluate the robustness of the learners against the noisy datasets. The learning experiments are performed with three complex events, namely the anchoredOrMoored, pilotOps and rendezVous. These events are chosen for the following reasons.

First, two of these events (pilotOps and rendezVous) are relational, therefore, their corresponding rule-based patterns involve more than one vessel. Moreover, they are the only definitions in the maritime complex event library that have been developed with this property. Performing learning experiments with these complex events were particularly useful, since dealing with relational knowledge is one of the important relative strengths of logic-based learning methods, as the ones that we investigate in this work. Finally, as we could see in the previous chapter, that the noise, generally, does not change much the recognized intervals, except the ultimate form of noise with distance threshold equals 1000, but these patterns are good for the experiments because they are pretty complex.

Learning non-relational patterns is also useful, however, most of the remaining complex event definitions have a number of drawbacks that renders learning them either ”too easy”, or impractical. Many complex events, such as withinArea, gap, stopped, etc. fall in the first category. These events are mostly used in the library as intermediate-level events, useful for defining higher-level ones, and their definitions are straightforward. On the other hand, several complex events with challenging definitions (from a learning perspective) such as movingSpeed, drifting, tuggingSpeed, trawlSpeed, trawlingMovement and sarSpeed, heavily involve learning numerical thresholds, a task that is not particularly well-handled by the logic-based learning algorithms that we study in this work. Extending the learning algorithms for efficiently dealing with such forms of threshold learning is an important direction for future work.

As a proof of concept, we experimented with one non-relational pattern learning, anchoredOrMoored, in addition to pilotOps and rendezVous (the relational ones).

To ease the learning effort only vessels that were close enough to each other (participating in a proximity event) were considered. This significantly reduces the ASP solver's grounding and solving cost during processing each data batch.

The following results are for four different values of c. Training times are in seconds and they are averages from the five folds.

Table 7.1: Learning results

Event	Noise Type	OLED		WOLED	
		F1-Score	Time	F1-Score	Time
anchoredOrMoored	no-noise	0.823	2834	0.878	3745
	noise with $c = 1000$	0.815	3188	0.872	3792
	noise with $c = 5000$	0.865	2945	0.767	3827
	noise with $c = 10000$	0.833	2986	0.858	3844
	noise with $c = 20000$	0.814	3022	0.877	3819
pilotOps	no-noise	0.743	3667	0.812	4212
	noise with $c = 1000$	0.742	3662	0.788	4143
	noise with $c = 5000$	0.712	3723	0.783	4202
	noise with $c = 10000$	0.712	3718	0.788	4218
	noise with $c = 20000$	0.713	3729	0.788	4308
rendezVous	no-noise	0.805	4122	0.886	5528
	noise with $c = 1000$	0.805	4224	0.878	5542
	noise with $c = 5000$	0.788	4332	0.878	5430
	noise with $c = 10000$	0.788	4348	0.860	5582
	noise with $c = 20000$	0.770	4345	0.871	5612

Observing, Table 7.1 we can observe the following.Firstly, F1-scores are comparable and relatively good, either with noise or without noise, which is what was expected, since noise injection did not affect the recognition of these complex events dramatically, as indicated by the RTEC results. Also, note that, even for $c = 1000$ were it is seem that noise influences much the anchoredOrMoored and pilotOps event, OLED and WOLED perform really well, which indicates that these learners are robust. However, and this is the important finding, WOLED out-performs OLED in almost all cases, even marginally, or significantly (e.g. rendezVous). There are cases where a learner performs slightly better with noise than without noise. We attribute that to the randomness related to the selection of training/testing set pairs.

8. CONCLUSIONS AND FUTURE WORK

8.1 Summary

In the current thesis, we evaluated the robustness of online ILP learners, OLED and WOLED, against noisy variations of the Maritime Dataset. In order to achieve that, we proposed a novel noise injection method that can be applied in a realistic setting, because it incorporates the uncertainty of the received Critical Point with the distance from the coast. Afterwards, we illustrated an optimization for that particular noise injection method in order to be applicable when we have a huge dataset. Then, in order to evaluate our method and the learners' robustness our experimental evaluation was split into two parts. Firstly, because the noise was injected into the Critical Points, it was mandatory to run RTEC in order to get the noisy HLE intervals. For that reason, we checked how our proposed method changes the clean HLE intervals, in order to know what to expect when we forward these datasets to learners. Moreover, we compared all recognized events with each other to determine how and why the HLE intervals are affected, which was seen that the noise affects most the HLEs that are defined as Simple Determined Fluents or have a very simple definition. Secondly, we executed OLED and WOLED with the clean datasets, in order to determine their robustness against the online generated noisy dataset. As it was expected from the previous experiment these learners were proved to be robust, even against the datasets that were subjected to the ultimate form of noise. Furthermore, not only it was shown that WOLED out-performed OLED in terms of F1-Score, but also it was proven more robust to noise. In conclusion, we illustrated that OLED and WOLED are robust and accurate about finding event definitions even in the noisy maritime dataset.

8.2 Future Work

The research paths for future work are many. Firstly, one bottleneck shown in our research was that the learners are not made to learn numerical threshold or interval length predicates, thus we could not experiment with events that involved them, consequently our range experimentation was limited to specific events. Moreover, the learning times of WOLED are greater than the ones of OLED mostly because of the probabilistic MAP inference that WOLED performs at every step, but WOLED yields a small improvement at predictive performance. Thus, someone can make the predictive performance even better or optimize the MAP inference. Finally, someone can focus on the evaluation of the robustness of OLED and WOLED against concept drift, which is the phenomenon of an event that changes its behavioral pattern through the course of time.

ABBREVIATIONS - ACRONYMS

ILP	Inductive Logic Programming
MMSI	Maritime Mobile Service Identity
OLED	Online Learning of Event Definitions
LfI	Learning from Interpretations
WoLED	Weighted Online Learning of Event Definitions
CAVIAR	Context Aware Vision Using Image-Based Active Recognition
AIS	Automatic Identification System
RTEC	Run-Time Event Calculus
HLE	High Level Event
LLE	Low Level Event
ASP	Answer Set Programming
TP	True Positive
FP	False Positive
FN	False Negative
OSL	Online Structure Learning
OSL_α	Online Structure Learning with Background Knowledge Axiomization
MAP	Maximum Aposteriori
MLN	Markov Logic Network
JTS	Java Topology Suite

BIBLIOGRAPHY

[1] Web-scale k-means clustering. pages 1177–1178, 01 2010.

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

[3] Hendrik Blockeel, Luc De Raedt, Nico Jacobs, and Bart Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3, 09 2000.

[4] Luc De Raedt. Logical and relational learning. page 1, 01 2008.

[5] Amit Dhurandhar and Alin Dobra. Distribution-free bounds for relational classification. *Knowledge and Information Systems*, 31:55–78, 04 2012.

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

[7] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Answer set solving in practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6:1–238, 12 2012.

[8] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[9] Tuyen Huynh and Raymond Mooney. In online structure learning for markov logic networks. pages 81–96, 09 2011.

[10] Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. Online learning of event definitions, 2016.

[11] Nikos Katzouris, Evangelos Michelioudakis, Alexander Artikis, and Georgios Paliouras. Online learning of weighted relational rules for complex event recognition. In *ECML/PKDD*, 2018.

[12] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 01 1985.

[13] Evangelos Michelioudakis, Anastasios Skarlatidis, Georgios Paliouras, and Alexander Artikis. Osla: Online structure learning using background knowledge axiomatization. volume 9851, pages 232–247, 09 2016.

[14] E.T. Mueller. Commonsense reasoning: An event calculus based approach: Second edition. *Commonsense Reasoning: An Event Calculus Based Approach: Second Edition*, pages 1–482, 01 2014.

[15] Katzouris N. and Artikis A. Woled: A tool for online learning weighted answer set rules for temporal reasoning under uncertainty. *17th Int. Conf. on Principles of Knowledge Representation & Reasoning (KR)*, 2020.

[16] ANASTASIOS SKARLATIDIS, ALEXANDER ARTIKIS, JASON FILIPPOU, and GEORGIOS PALIOURAS. A probabilistic logic programming event calculus. *Theory and Practice of Logic Programming*, 15(2):213–245, May 2014.