

Improving DL-Learner on a Malware Detection Use Case

Tomáš Bisták¹, Peter Švec², Ján Kľuka¹, Alexander Šimko¹, Štefan Balogh² and Martin Homola¹

¹Department of Applied Informatics, Faculty of Mathematics, Physics and Informatics, Comenius University, Mlynská dolina, 842 48 Bratislava, Slovakia

²Institute of Computer Science and Mathematics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Ilkovičova 3, 812 19 Bratislava, Slovakia

Abstract


Automation of malware characterization has become increasingly important for early malware detection over the past decades. Since it is crucial to be able to perform malware detection transparently, explainable machine-learning methods may be considered particularly suitable for this application. In our previous work, we thus employed algorithms for learning description-logics concept expressions to obtain structured characterizations of malicious software. The results of our initial experiments, however, lead us to several inconsistencies in the behaviour of the studied algorithms included in DL-Learner, a widely adopted framework for concept learning. Hence, in this work, we make a couple of corrections to DL-Learner, along with a few further enhancements, and repeat the experiments to empirically assess the effects of our changes. The outcomes show that the modified algorithms can mark an improvement in terms of the predictability of their behaviour as well as evaluation metrics.


Keywords


DL-Learner, concept learning, malware detection, explainability


1. Introduction


Malware detection is an active research area of great significance within cybersecurity. Over the years, many different approaches to this problem have been devised, some of which, mainly those relying on analyses performed by malware experts, are currently used in practice [1]. The traditional methods requiring extensive human input are, however, gradually being phased out by techniques based on machine learning due to the ever-increasing amounts of new malware [2, 3, 4]. Despite the impressive progress in this direction, the malware community still recognizes a few weaknesses of these approaches, with one of the most severe being the lack of interpretability. Much of the recent research thus focused on harnessing various methods that are inherently explainable [5, 6, 7, 8, 9], including the use of concept learning in description

 DL 2023: 36th International Workshop on Description Logics, September 2–4, 2023, Rhodes, Greece

 bistak5@uniba.sk (T. Bisták); peter.svec1@stuba.sk (P. Švec); kluca@fmph.uniba.sk (J. Kľuka); alexander.simko@fmph.uniba.sk (A. Šimko); stefan.balogh@stuba.sk (Š. Balogh); homola@fmph.uniba.sk (M. Homola)

 0009-0005-8715-0574 (T. Bisták); 0000-0002-8315-5301 (P. Švec); 0000-0002-3406-3574 (J. Kľuka); 0009-0009-5725-1349 (A. Šimko); 0000-0003-0634-9476 (Š. Balogh); 0000-0001-6384-9771 (M. Homola)

 © 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

logics we proposed in [10].

Concept learning allows us to induce malware characterizations in the form of concept expressions over the vocabulary of a suitable ontology only from input expressions describing a selected sample of software. Our previous work [10], in which we applied DL-Learner [11] (widely considered a state-of-the-art concept learning tool) on data from the EMBER dataset [12] supplemented by a specifically designed ontology [13], showed that the resulting concepts exhibit decent levels of readability and interpretability from an expert’s point of view.

However, the initial experiments revealed several flaws both in the theoretical bases and in the implementation of the employed learning algorithms, impacting their correctness and completeness. Apart from that, we also identified a few opportunities to improve the efficiency of the studied DL-Learner algorithms.

In this work, we present the detected shortcomings and modify the official distribution of DL-Learner to address them. We then assess the effects of our changes by comparing the results from the modified algorithms to those obtained with the original implementation. The outcomes indicate that the altered algorithms achieve more predictable results in terms of learned concept descriptions and even slightly outperform the original implementation in the evaluation metrics on average.

2. Concept Learning and DL-Learner

2.1. Concept-Learning Problem

Recall that the languages of description logics [14, 15] are defined over *vocabularies* $N = (N_I, N_C, N_R)$ consisting, respectively, of mutually disjoint sets of *individual*, *concept*, and *role names*. A description logic (DL) \mathcal{L} defines the sets of *role expressions* \mathbf{R} and *concepts (descriptions, concept expressions)* \mathbf{C} over N as certain subsets of expressions generated, respectively, by grammars

$$\begin{aligned} R &::= r \mid r^- \\ C &::= A \mid \{a\} \mid \top \mid \perp \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \mid \geq n R.C \mid \leq n R.C \end{aligned}$$

for $r \in N_R$, $A \in N_C$, $a \in N_I$, and positive integers n . A *knowledge base* (KB) \mathcal{K} in \mathcal{L} over N is a finite set of *role and concept inclusion axioms* ($R_1 \sqsubseteq R_2$, $C_1 \sqsubseteq C_2$), and *individual and role assertions* ($C(a)$, $R(a_1, a_2)$), subject to the syntactical restrictions of \mathcal{L} . We assume the usual DL semantics [14, 15], in particular the *entailment* $\mathcal{K} \models X$ of an axiom or assertion X from a KB \mathcal{K} . A concept C *subsumes* a concept D ($C \sqsubseteq D$) if $\emptyset \models C \sqsubseteq D$.

The problem of learning concept expressions, which we aim to use to characterize malware, is defined as follows [16]:

Definition 1 (Concept-Learning Problem). *Let \mathcal{L} be a DL and \mathcal{K} be a knowledge base in \mathcal{L} over a vocabulary (N_I, N_C, N_R) . Let $E^+ \subseteq N_I$ be a set of positive examples and $E^- \subseteq N_I$ be a set of negative examples, with $E^+ \neq \emptyset$ and $E^+ \cap E^- = \emptyset$.*

Given \mathcal{K} , E^+ , and E^- , find a description $C \in \mathbf{C}$ satisfying both of the following conditions:

$$\mathcal{K} \models C(a) \text{ for all } a \in E^+, \tag{1}$$

$$\mathcal{K} \not\models C(a) \text{ for all } a \in E^-. \tag{2}$$

For any $a \in E^+ \cup E^-$ and $C \in \mathbf{C}$, we say that C covers a if $\mathcal{K} \models C(a)$ holds.

The goal of concept learning is thus to find a description of a target class (in our case, malware) given some examples of individuals belonging to this class, as well as some counterexamples. Note that Definition 1 is adjusted to the closed-world assumption (CWA), under which we decided to work since all the DL-Learner algorithms we examine here were designed primarily for CWA scenarios.

2.2. Learning Concepts with Refinement Operators

As a concept-learning problem is considered to be solved once an appropriate description is found, one may cast this problem as a search problem where we search the space of all valid concept expressions ordered by the standard DL relation of subsumption [16]. Functions producing sets of successors of concept expressions w.r.t. subsumption are called *refinement operators*.

Definition 2 (Refinement Operator). *Let \mathbf{C} be the set of all descriptions in a DL \mathcal{L} over some vocabulary. A mapping $\rho : \mathbf{C} \rightarrow 2^{\mathbf{C}}$ is a **downward (upward) refinement operator** in the quasi-ordered space $(\mathbf{C}, \sqsubseteq)$ if $D \in \rho(C)$ implies $D \sqsubseteq C$ ($C \sqsubseteq D$) for all $C \in \mathbf{C}$. If $D \in \rho(C)$, the concept D is a **refinement** of C , more specifically, a **specialization (generalization)** of C .*

For example, if we let \rightsquigarrow denote a single refinement operation, one branch of a search tree (*refinement chain*) induced by a downward refinement operator may have the following form:

$$\top \rightsquigarrow \forall r. \top \rightsquigarrow \forall r. C \rightsquigarrow (\forall r. C) \sqcap D. \quad (3)$$

Usually, the process of downward refinement starts from the top concept.

2.3. Examined DL-Learner Algorithms

In our experiments, we evaluate the performance of four concept-learning algorithms implemented in DL-Learner [11], namely, OCEL [17], CELOE [18], PARCEL [19], and SPACEL [20]. All of them use the downward refinement operator called simply ρ^1 [17] in combination with additional heuristics to guide the search. As the set of refinements is not finite for most concepts in the case of ρ , these algorithms limit the length of refinements and proceed iteratively, revisiting any of the already searched concepts whenever they find it promising based on its heuristic score and gradually asking for longer refinements.

OCEL was the first concept-learning algorithm based on a refinement operator. CELOE was later introduced as a variation of OCEL attracted to shorter concepts following the principle of Occam's razor, i.e., that simpler descriptions may generalize better since they fit the training conditions more loosely. Both OCEL and CELOE search for the final description as a whole.

PARCEL and SPACEL parallelize the learning process by constructing descriptions in the form of a disjunction and searching for the disjuncts (*partial definitions*) in parallel. While PARCEL infers the partial definitions only from the characteristics of positive examples, SPACEL works also with the found partial descriptions of negative examples by combining their negations with otherwise mediocre expressions to create partial definitions.

¹For brevity, we do not include the definition of ρ , albeit we refer to it in Section 4.3.1.

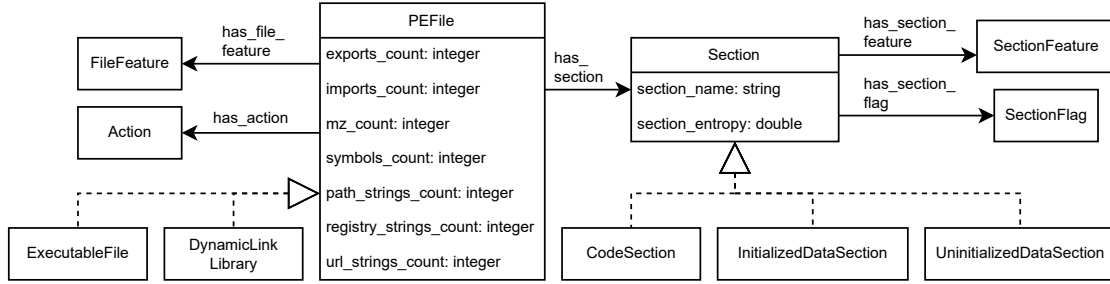


Figure 1: Core classes of the PE malware ontology and their properties

3. Ontology and Dataset

3.1. EMBER Dataset

Elastic Malware Benchmark for Empowering Researchers (EMBER) [12] is a dataset for training static malware detection models. It contains structured data entries of 1.1 million Windows portable executable files (PE files). Out of them, 800,000 entries are labelled as either malicious or benign (400,000 each), and 300,000 entries are left unlabeled.

Each data entry is a JSON object with properties such as: the corresponding file’s size; the number of imported and exported functions; the target architecture; information about the file’s sections, e.g., section names, content type, access rights; a list of imported functions per DLL; a list of exported functions; various histograms and statistics, and more.

3.2. PE Malware Ontology

In order to use EMBER and similar data sources in concept learning, we need to capture them semantically via an ontology. To this end, we designed the *PE Malware Ontology*, which is a light-weight OWL 2 ontology expressed within DL-Lite_{core}(D), i.e., the OWL 2 QL profile. In total, the ontology comprises 195 classes, 6 object properties, and 10 data properties. We are giving a brief overview of the ontology here and refer the reader interested in more details to our technical report [13].

Focusing on interpretability, the ontology represents the PE file structure and mostly qualitative features that make sense from a malware expert’s point of view. The core classes and properties of the ontology are shown in Figure 1. Each dataset sample is described by an instance of the *PEFile* class, and structurally consists of sections expressed by the *Section* class. *PEFiles* and *Sections* are further classified by their content using subclasses.

A few selected, easily interpretable quantitative features are assigned to *PEFiles* and *Sections* via data properties (e.g., *imports_count* or *section_entropy*). Section names are also included since they are often unusual in malware samples. We disregarded many quantitative features such as file sizes, byte histograms, or linker versions. Although some of them are statistically useful in ML-based malware detection [21], they are also difficult to interpret and could potentially lead to the trained classifier’s susceptibility to rather trivial adversarial attacks [22].

A total of 15 relevant qualitative features of PE files and 3 features of sections are represented by subclasses of `FileFeature` and `SectionFeature`, respectively, and linked to `PEFile` and `Section` instances by object properties. Instances of `SectionFlag` represent the section in-memory permission flags, which may indicate, e.g., self-modifying code.

Important clues about the possible behaviour of a running PE file are provided by the functions it imports from standard system libraries. We abstracted these functions into possible actions expressed by the `Action` class and linked to `PEFiles`. Its subclasses represent particular actions from the MAEC Vocabulary of malware actions [23] with minor adjustments detailed in the technical report [13]. In order to aid generalization, we included additional 17 abstract action classes.

3.3. Fractional Datasets

To facilitate reproduction of experimental results, we created a collection of RDF datasets based on the PE Malware Ontology and EMBER data. The ontology and the datasets are available in our GitHub repository². The complete dataset (named `dataset_1_800000.owl`) contains all 800k of labelled samples from the EMBER dataset. Since concept learning is computationally expensive, we generated fractional datasets of 1k, 10k, and 100k of randomly selected samples. Ten variants were produced for each size (`dataset_N_size.owl`, $N \in \{1, \dots, 10\}$) to help researchers compensate for selection bias. The 1k datasets contain 500 malicious (positive) and 500 benign (negative) samples, which amounts to 5.8k individuals and 63k axioms (of which 5.8k are class, 34k object-property, and 16k data-property assertions, 1.5k are ontology axioms, and 5.8k are non-logical axioms) on average.

4. Applying DL-Learner

4.1. Methodology and Experiments

Due to the computational complexity of concept learning algorithms, we only carried out our experiments on the 1k datasets from our suite but in two phases. In the first phase, we calibrated several hyper-parameters (see below) on the dataset `dataset_8_1000.owl` selected at random. In the second phase, the best configurations were validated on the 1k datasets 1-5.

The calibrated parameters were noise, the use of `hasValue` and negation constructors, the *some-only* rule, and a cardinality limit. Noise is a common parameter in concept-learning algorithms that can be used as a termination criterion (specifying the minimum acceptable accuracy), as a part of heuristics, or as an instrument for discarding weak concepts during learning. The `hasValue` constructor allows the refinement operator to generate descriptions of the form $\exists r.\{a\}$, with r an object property and a an individual. The negation constructor enables the generation of class expressions of the form $\neg C$, for any atomic concept C . If activated, the *some-only* option ensures that universal restrictions on any object property r are used only in conjunction with a minimum-cardinality or existential restriction on r . The cardinality limit represents the maximum cardinality n with which the qualified number restrictions $\leq n r.C$,

²<https://github.com/orbis-security/pe-malware-ontology>

Table 1Calibration results on test folds (*avg ± std*).

Algorithm	Accuracy	Precision	Recall	FP Rate	F1
PARCEL	0.76 ± 0.04	0.76 ± 0.07	0.76 ± 0.04	0.24 ± 0.09	0.76 ± 0.03
PARCEL fixed	0.79 ± 0.01	0.82 ± 0.02	0.73 ± 0.05	0.15 ± 0.03	0.77 ± 0.02
SPACEL	0.77 ± 0.03	0.76 ± 0.03	0.79 ± 0.02	0.24 ± 0.04	0.77 ± 0.02
SPACEL fixed	0.76 ± 0.03	0.76 ± 0.03	0.77 ± 0.04	0.24 ± 0.04	0.76 ± 0.03
OCEL	0.72 ± 0.02	0.80 ± 0.03	0.59 ± 0.03	0.14 ± 0.02	0.68 ± 0.03
OCEL fixed	0.75 ± 0.02	0.77 ± 0.03	0.72 ± 0.04	0.22 ± 0.05	0.74 ± 0.02
CELOE	0.71 ± 0.03	0.70 ± 0.03	0.74 ± 0.03	0.30 ± 0.04	0.72 ± 0.03
CELOE fixed	0.70 ± 0.02	0.71 ± 0.05	0.68 ± 0.10	0.27 ± 0.11	0.69 ± 0.03

Table 2Validation results averaged across the five validation datasets (*avg ± std*).

Algorithm	Accuracy	Precision	Recall	FP Rate	F1
PARCEL	0.72 ± 0.01	0.71 ± 0.01	0.72 ± 0.02	0.28 ± 0.01	0.72 ± 0.01
PARCEL fixed	0.73 ± 0.02	0.76 ± 0.02	0.66 ± 0.03	0.20 ± 0.01	0.71 ± 0.03
SPACEL	0.72 ± 0.01	0.72 ± 0.00	0.73 ± 0.02	0.27 ± 0.00	0.72 ± 0.01
SPACEL fixed	0.72 ± 0.03	0.72 ± 0.02	0.73 ± 0.04	0.28 ± 0.02	0.72 ± 0.03
OCEL	0.69 ± 0.01	0.68 ± 0.05	0.74 ± 0.10	0.35 ± 0.12	0.70 ± 0.02
OCEL fixed	0.73 ± 0.01	0.72 ± 0.01	0.74 ± 0.04	0.27 ± 0.02	0.73 ± 0.02
CELOE	0.68 ± 0.01	0.65 ± 0.03	0.77 ± 0.05	0.40 ± 0.07	0.70 ± 0.01
CELOE fixed	0.70 ± 0.01	0.68 ± 0.02	0.79 ± 0.06	0.38 ± 0.06	0.72 ± 0.02

$\geq nr.C$, and $=nr.C$ (i.e., $\leq nr.C \sqcap \geq nr.C$) can be produced, where r is an object property and C is a class expression.

For evaluation, we used the k -fold cross-validation technique, with $k = 5$, as it works quite well with smaller and limited datasets [24]. The performance itself was assessed via standard metrics – accuracy, precision, recall, F1 score, and false-positive (FP) rate [25]. We experimented on a machine with an 18-core Intel Core i9-10980XE CPU, 256 GB of RAM and Debian 5.10-140-1. The maximum training time in each iteration of the 5-fold cross-validation was limited to 24 and 2 hours of *user time* (the time the CPU spends executing user-space code) for the parallel (PARCEL and SPACEL) and the non-parallel (OCEL and CELOE) algorithms, respectively. Since we configured the parallel algorithms to use 12 working threads and the non-parallel algorithms run in a single execution thread, these limits correspond to approx. 2 hours of real-world time.

4.2. Out-of-the-Box Results

The results from the calibration phase for the best hyper-parameter settings are provided in Table 1 in rows without the “fixed” suffix. As we can see, the most promising descriptions were obtained using PARCEL and SPACEL, which achieved an F1 score of 0.76 and 0.77, respectively. During the calibration phase, we also noticed the following interesting phenomenon. While disabling the `hasValue` constructor should reduce the search space and help the algorithms to find solutions more efficiently due to the nature of our ontology (because for any valuable

concept $\exists r.\{a\}$, there always exists such a concept $\exists r.C$, where C is not a nominal, that is equivalent with $\exists r.\{a\}$ under the CWA in our scenario), it merely substantially degraded the performance of both parallel algorithms w.r.t. the studied metrics (e.g., the F1 score dropped from 0.71 to 0.60 for some PARCEL configurations). In the case of the non-parallel algorithms, disabling the `hasValue` constructor caused no changes in terms of the metrics at all.

The validation results, shown in Table 2, generally confirmed the quality of the configurations selected in the calibration phase, although the accuracy and the F1 score were mostly lower. An indicative class expression generated by OCEL, with a test accuracy of 0.75, is given in (4).

$$\begin{aligned} \text{ExecutableFile} \sqcap \exists \text{has_section}.\exists \text{has_section_feature}.\text{NonstandardSectionName} \\ \sqcap \exists \text{has_section}.\exists \text{has_section_flag}.\{\text{writable}\} \end{aligned} \quad (4)$$

4.3. Issues, Corrections, and Enhancements

After performing the initial tests, we detected a couple of flaws in the behaviour of the employed learning algorithms. To resolve these issues and further improve the performance of the learners, we slightly modified the theoretical definition of the ρ refinement operator, as well as its implementation and the implementation of the examined learning algorithms in the official distribution of DL-Learner [26]. The updated version of DL-Learner is available in our GitHub repository³. In this section, we discuss the most pivotal changes.

4.3.1. Refinement Operator ρ

At-Most Restrictions. After noticing some inexplicable differences in the accuracy of two logically equivalent descriptions reported by OCEL, we found out that the ρ refinement operator was handling at-most restrictions inappropriately. More specifically, we realized that the operation

$$\leq n r.D \rightsquigarrow \leq n r.E, \quad (5)$$

for any concept D , any simple role r , $n \in \mathbb{N}$, and $E \in \rho_{\text{ar}(r)}(D)$, from the extended definition of ρ (see Section 5.4 in [17]) actually generates upward refinements if we assume that $E \sqsubseteq D$, what we can expect to be satisfied for a downward refinement operator like ρ .

To cope with this problem, we redefined this operation as follows⁴:

$$\leq n r.D \rightsquigarrow \leq n r.\overline{E}, \quad (6)$$

where $E \in \rho_{\text{ar}(r)}(\overline{D})$, i.e., we inverted the direction in which the constrained concept is refined (from downwards to upwards). It can be easily shown by the principle of structural induction for a selected DL \mathcal{L} that when we change (5) to (6) in the definition of ρ , the ρ operator becomes a downward refinement operator (not considering the refinement of universally quantified roles and double concrete roles, see the note below). The key is to leverage the facts that the inverse of (5) is an operation of specialization and that if $E \sqsubseteq \overline{D}$ (which is a consequence of the premise that $E \in \rho_{\text{ar}(r)}(\overline{D})$), then $D \sqsubseteq \overline{E}$.

³<https://github.com/mousetom-sk/DL-Learner/tree/v3>

⁴We use \overline{C} to denote the negation of a concept C in the negation normal form [14, 15].

Of course, this modification also requires the refinement process of at-most restrictions to start from the concept $\leq n r. \perp$ instead of $\leq n r. \top$ as in the original definition of ρ [17].

Note that there are similar issues with the refinement of roles inside universal quantifications, but we decided not to address them since all object properties in the PE Malware Ontology are direct subproperties of `owl:topObjectProperty`. Double concrete roles are also refined in the opposite direction in [17], nonetheless, the most recent version of DL-Learner already implements the corrected definition of these refinement operations.

Negations and Universal Quantification. Studying the definition of the ρ refinement operator more carefully, we later detected at least two other places for potential improvement.

Firstly, we prevented the operator from generating refinements of the form $\neg C$, where $C \in \mathbf{C}$ is a superconcept of the domain in which the refinement is performed, e.g., the range of a role. Clearly, this does not decrease the expressive power of the operator, but it helps to reduce the number of nodes in the search tree.

Secondly, we decided to shorten the refinement path from the top concept to the universal quantification concepts of the following kind: $\forall r. \perp$. Originally, a concept $\forall r. C$, with a most specific concept name $C \in N_C$, i.e., one for which no $D \in N_C$ exists such that $D \sqsubseteq C$, had to be reached in order to produce $\forall r. \perp$. However, if such universal quantifications are not promising enough, $\forall r. \perp$ is never introduced during refinement, even though it bears a slightly different meaning than other universal quantifications, e.g., in our context, $\forall \text{has_section_feature}. \perp$ represents all sections with none of the special features. Contrarily, when a concept of the form $\forall r. \perp$ is a refinement of each concept $\forall r. C$, where C is most specific, it could also quite unnecessarily appear multiple times in the search tree. To address these issues, we redefined the ρ operator in a way that makes $\forall r. \perp$ exclusively a direct refinement of $\forall r. \top$.

4.3.2. Cardinality Constraints in Closed-World Reasoner

Even after redefining the ρ operator as suggested in Section 4.3.1, OCEL and CELOE could not agree on the accuracy of concepts containing at-most restrictions. We discovered that this issue was caused by DL-Learner's default closed-world reasoner, which we employed, since it did not always correctly perform instance checking for concepts with cardinality constraints.

As this faulty behaviour stemmed purely from programming errors, the problem was simply fixed by their rectification.

4.3.3. Heuristic in OCEL

One undocumented component of OCEL's heuristic function was originally designed to motivate the refinement of concepts containing $\forall r. \top$ or $\leq n r. \top$ by adding a bonus to their heuristic score. The main reason behind this step was that $\forall r. \top$ and $\leq n r. \top$ provide no or just a little information, respectively. Furthermore, the former is also the only way to all universal restrictions on the role r and the latter had to be generated to obtain any at-most restriction on r before.

The implementation of the calculation of this bonus was, however, non-deterministic and the adjustment to the heuristic score did not reflect how many times any of the above two

expressions appeared in the evaluated concept. Moreover, after the changes to the refinement rule for at-most-restrictions, we should favour $\leq n r. \perp$ rather than $\leq n r. \top$, as a path to any at-most restriction on r now leads through $\leq n r. \perp$.

Hence, we modified this heuristic in order to ensure its determinism and that the score correction grows linearly with the increasing number of occurrences of the expressions in question. We replaced the preference for the refinement of concepts containing $\leq n r. \perp$ with the support of those with $\leq n r. \top$ as well.

4.3.4. PARCEL and SPACEL

Same-Length Refinements. Investigating the implementation of the learning process, we noticed that neither of the parallel algorithms was able to reach refinements whose length was equal to the length of their direct predecessor, such as $\forall r. C$, with $C \in N_C$.

We resolved this issue by allowing PARCEL and SPACEL to accept the direct refinements which are of the same length as the refined concept.

Accuracy and Coverage Computation in PARCEL. When determining the accuracy of a concept expression and its coverage of positive examples for the purposes of heuristics, PARCEL considered solely the positive examples that were not yet covered by the hitherto learned partial definitions as the universe of all positive examples. We assume that the rationale behind this exclusion of the already covered positive examples was to guide the search towards the expressions that can expand the overall coverage of positive examples the most.

A downside of such a biased definition of accuracy/coverage is that it takes into account not only the global quality of a concept but also when it is evaluated. To eliminate the effects of the second aspect on a later choice of concepts for refinement (based on accuracy and coverage), we would have to update the measurements for all previously searched concepts whenever a new partial definition is found. Nevertheless, PARCEL computes a concept's accuracy/coverage merely upon its discovery as this operation requires time-demanding instance checking.

We therefore re-implemented PARCEL so that it utilizes the standard definitions of accuracy and coverage, while introducing the following two rules to draw the algorithm's attention to more promising concepts. Firstly, only the concepts that still cover at least one of the yet uncovered positive examples may be further refined. Secondly, a description can be marked as a partial definition only if it covers strictly more of the uncovered positive than of the uncovered negative examples.

Acceleration of Accuracy and Coverage Computation. Since ρ is a downward refinement operator, and thus generates subconcepts, we know that to evaluate the accuracy and the coverage of a given concept's refinement, it is enough to find out which of the positive and the negative examples covered by the original concept are also covered by that particular refinement. We leveraged this fact to speed up the computation of accuracy and coverage in PARCEL and SPACEL by storing the information on what sets of positive and negative examples each concept expression covers in the corresponding node of the search tree. To cope with the induced increase in the size of the search tree, we were also forced to compress the representation of the sets of covered examples by mapping individuals to integers.

We have to note that this modification was inspired by the implementation of OCEL, which uses a similar technique out of the box.

Search Tree Organization. The last change we discuss is more technical than those mentioned above, yet we decided to present it due to its immense impact despite its simplicity.

Both PARCEL and SPACEL use Java’s `ConcurrentSkipListSet` to keep all the nodes of the search tree in the order determined by the heuristic scores assigned to the concepts they embody. Originally, the nodes with the best concepts according to the heuristic were located at the end of this ordered set, so PARCEL and SPACEL were always polling its last element to obtain the most appropriate concept for refinement.

However, the official documentation for `ConcurrentSkipListSet` [27] stresses that the constituent parts of the objects of this type can be accessed faster via ascending iterators than via descending ones, indicating that polling the first element may take less time than removing the last. We confirmed this hypothesis through dedicated testing, during which retrieving the first element proved to be almost two times faster. Consequently, we reversed the order of nodes in the search tree and reprogrammed the algorithms to poll the first element in place of the last.

It is also important to state that the order of elements does not affect the insertion time.

4.4. Improved Results

After implementing the corrections and enhancements mentioned in Section 4.3, we repeated the experimentation process described in Section 4.1 with the updated DL-Learner. The results from the calibration and the validation phase are displayed in Tables 1 and 2, respectively (the rows labelled as “fixed”).

As these tables indicate, PARCEL now reached a noticeably lower FP rate than in the out-of-the-box experiments with roughly the same value of F1 score. CELOE marked an improvement in the FP rate as well, while OCEL achieved a higher F1 score. Unlike in the previous experiments, the corrected algorithms performed better when we disabled the `hasValue` constructor as expected. For instance, the F1 score for SPACEL increased from 0.69 to 0.76 purely due to this configuration change. Regardless of the hyper-parameter settings, the parallel algorithms were also able to search larger portions of the space. Specifically, PARCEL produced around 8 million descriptions in 24 hours of user time, which is approx. two times more than before. Similarly, the number of descriptions generated by SPACEL increased from 1.5 million to 7 million. The amount of class expressions discovered by the non-parallel algorithms either remained identical or slightly lowered. Nonetheless, these algorithms traversed the search space more efficiently here than in the first set of experiments, as the quality of their descriptions increased in various respects, e.g., the below-discussed level of detail.

To compare the algorithms w.r.t. the learned descriptions, we also share the expression (7) generated by OCEL on the same fold as (4) and during the same training period, although with a bit different hyper-parameter settings, e.g., for `hasValue`. Reaching a test accuracy of 0.77, the expression (7) demonstrates that OCEL was able to dive deeper into the search tree to learn more detailed and accurate descriptions after our modifications.

$$\begin{aligned}
& \text{ExecutableFile} \sqcap \exists \text{has_file_feature.} (\text{MultipleExecutableSections} \sqcup \text{NonstandardMZ}) \\
& \sqcap \exists \text{has_section.} \exists \text{has_section_flag.} \text{Writable} \\
& \sqcap \leq 1 \text{ has_action.} (\text{AcceptSocketConnection} \sqcup \text{DirectoryHandling} \\
& \quad \sqcup \text{EnumerateThreads} \sqcup \text{GetCurrentDirectory} \sqcup \text{OpenMutex})
\end{aligned} \tag{7}$$

5. Conclusions

In this study, we applied DL-Learner on a dataset from a malware-detection use case with the aim to learn concept descriptions that can be used as interpretable characterizations of the malware samples present in the dataset. We discovered and corrected a number of issues in the implementation of DL-Learner, which lead to an improvement in the obtained results, as our evaluation demonstrated.

Nevertheless, our case studies are only preliminary, since we have been able to process just a tiny fraction of the available data so far. Learning from our larger fractional datasets may potentially increase the quality of the found concepts and their evaluation metrics. As this is computationally expensive, we need to search for more sophisticated strategies for breaking down the data (e.g., based on particular malware families) and then combining the results. We would also like to explore and compare other concept-learning systems in the context of malware characterization in the future [28, 29, 30], including those capable of learning fuzzy concept descriptions [31, 32, 33]. In addition, while the obtained concepts proved to be fairly interpretable, we would like to conduct a more rigorous study of their usefulness with malware experts.

Last but not least, the malware detection area offers an interesting application domain with its own specifics, and above all, with vast quantities of real-world data that may be readily explored as a new use case [34] not just for concept-learning but also for the broader area of structured machine learning [35].

Acknowledgments

The authors would like to thank Umberto Straccia, Claudia d’Amato, and others who provided insights leading to this work. This research was sponsored by the Slovak Republic under the grant APVV-19-0220 (ORBIS) and by the EU under the H2020 grant no. 952215 (TAILOR).

References

- [1] A. Souri, R. Hosseini, A state-of-the-art survey of malware detection approaches using data mining techniques, *Hum.-Centric Comput. Inf. Sci.* 8 (2018) 1–22.
- [2] D. Ucci, L. Aniello, R. Baldoni, Survey of machine learning techniques for malware analysis, *Computers & Security* 81 (2019) 123–147.

- [3] D. Gibert, C. Mateu, J. Planes, The rise of machine learning for detection and classification of malware: Research developments, trends and challenges, *Journal of Network and Computer Applications* 153 (2020) 102526.
- [4] K. Shaukat, S. Luo, V. Varadharajan, I. A. Hameed, M. Xu, A survey on machine learning techniques for cyber security in the last decade, *IEEE Access* 8 (2020) 222310–222354.
- [5] G. Iadarola, F. Martinelli, F. Mercaldo, A. Santone, Towards an interpretable deep learning model for mobile malware detection and family identification, *Computers & Security* 105 (2021) 102198.
- [6] B. Marais, T. Quertier, C. Chesneau, Malware analysis with artificial intelligence and a particular attention on results interpretability, in: *Distributed Computing and Artificial Intelligence, Volume 1: 18th International Conference, DCAI 2021, Salamanca, Spain, 6-8 October 2021, volume 327 of LNNS*, Springer, 2021, pp. 43–55.
- [7] A. Amich, B. Eshete, Explanation-guided diagnosis of machine learning evasion attacks, in: *Security and Privacy in Communication Networks – 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6-9, 2021, Proceedings, Part I, volume 398 of LNICST*, Springer, 2021, pp. 207–228.
- [8] A. Mills, T. Spyridopoulos, P. Legg, Efficient and interpretable real-time malware detection using random-forest, in: *2019 International conference on cyber situational awareness, data analytics and assessment (Cyber SA)*, IEEE, 2019, pp. 1–8.
- [9] J. Dolejš, M. Jureček, Interpretability of machine learning-based results of malware detection using a set of rules, in: M. Stamp, C. Aaron Visaggio, F. Mercaldo, F. Di Troia (Eds.), *Artificial Intelligence for Cybersecurity*, Springer, Cham, 2022, pp. 107–136.
- [10] P. Švec, Š. Balogh, M. Homola, Experimental evaluation of description logic concept learning algorithms for static malware detection., in: *ICISSP, 2021*, pp. 792–799.
- [11] J. Lehmann, DL-learner: Learning concepts in description logics, *The Journal of Machine Learning Research* 10 (2009) 2639–2642. doi:10.5555/1577069.1755874.
- [12] H. S. Anderson, P. Roth, EMBER: An open dataset for training static PE malware machine learning models, *arXiv preprint arXiv:1804.04637* (2018).
- [13] P. Švec, Š. Balogh, M. Homola, J. Klůka, Knowledge-based dataset for training PE malware detection models, *arXiv preprint arXiv:2301.00153* (2022).
- [14] F. Baader, I. Horrocks, C. Lutz, U. Sattler, *An Introduction to Description Logic*, Cambridge University Press, 2017.
- [15] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.
- [16] J. Lehmann, P. Hitzler, Foundations of refinement operators for description logics, in: *Inductive Logic Programming*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 161–174.
- [17] J. Lehmann, P. Hitzler, Concept learning in description logics using refinement operators, *Machine Learning* 78 (2009) 203–250.
- [18] J. Lehmann, S. Auer, L. Bühmann, S. Tramp, Class expression learning for ontology engineering, *Journal of Web Semantics* 9 (2011) 71–81.
- [19] A. C. Tran, J. Dietrich, H. W. Guesgen, S. Marsland, An approach to parallel class expression learning, in: *Rules on the Web: Research and Applications*, Springer, Berlin, Heidelberg,

2012, pp. 302–316.

- [20] A. C. Tran, J. Dietrich, H. W. Guesgen, S. R. Marsland, Two-way parallel class expression learning, in: Asian Conference on Machine Learning, 2012.
- [21] Y. Oyama, T. Miyashita, H. Kokubo, Identifying useful features for malware detection in the ember dataset, in: 2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW), 2019, pp. 360–366.
- [22] O. Suciuc, S. E. Coull, J. Johns, Exploring adversarial examples in malware detection, in: 2019 IEEE Security and Privacy Workshops (SPW), IEEE, 2019, pp. 8–14.
- [23] MITRE Corp., MAEC™ 5.0 specification. Vocabularies, 2017. URL: https://maecproject.github.io/releases/5.0/MAEC_Vocabularies_Specification.pdf, [Online; accessed 2022-05-15].
- [24] O. Maimon, L. Rokach (Eds.), The Data Mining and Knowledge Discovery Handbook, Springer, 2005.
- [25] T. Fawcett, An introduction to ROC analysis, Pattern recognition letters 27 (2006) 861–874.
- [26] J. Lehmann, et al., DL-Learner, 2021. URL: <https://dl-learner.org>, [Version 1.5.0].
- [27] Oracle Corp., ConcurrentSkipListSet (Java Platform SE 8), 2023. URL: <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentSkipListSet.html>, [Online; accessed 2023-05-28].
- [28] N. Fanizzi, G. Rizzo, C. d’Amato, F. Esposito, DLFOil: Class expression learning revisited, in: European Knowledge Acquisition Workshop, Springer, 2018, pp. 98–113.
- [29] G. Rizzo, N. Fanizzi, C. d’Amato, Class expression induction as concept space exploration: From DL-FOIL to DL-FOCL, Future Generation Computer Systems 108 (2020) 256–272.
- [30] S. Heindorf, L. Blübaum, N. Düsterhus, T. Werner, V. N. Golani, C. Demir, A. N. Ngomo, Evolearner: Learning description logics with evolutionary algorithms, in: WWW ’22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022, ACM, 2022, pp. 818–828.
- [31] U. Straccia, M. Mucci, pFOIL-DL: Learning (fuzzy) EL concept descriptions from crisp OWL data using a probabilistic ensemble estimation, in: Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015, pp. 345–352.
- [32] F. A. Cardillo, U. Straccia, Fuzzy OWL-Boost: Learning fuzzy concept inclusions via real-valued boosting, Fuzzy Sets Syst. 438 (2022) 164–186.
- [33] F. A. Cardillo, F. Debole, U. Straccia, PN-OWL: A two stage algorithm to learn fuzzy concept inclusions from OWL ontologies, arXiv preprint arXiv:2303.07192 (2023). doi:10.48550/ARXIV.2303.07192.
- [34] P. Švec, T. Bisták, M. Homola, Štefan Balogh, J. Kluka, A. Šimko, Towards explainable malware detection with structured machine learning (Extended abstract), in: The Fourth Workshop on Explainable Logic-Based Knowledge Representation (XLoKR 2023), 2023.
- [35] P. Westphal, L. Bühmann, S. Bin, H. Jabeen, J. Lehmann, SML-Bench—A benchmarking framework for structured machine learning, Semantic Web 10 (2019) 231–245.