Towards Robust Neurosymbolic Relational Learning

Thais Luca
Systems Engineering and Computer Science, COPPE
Universidade Federal do Rio de Janeiro
Rio de Janeiro, RJ, Brazil
tluca@cos.ufrj.br

Aline Paes
Institute of Computing
Universidade Federal Fluminense
Niteroi, RJ, Brazil
alinepaes@ic.uff.br

Gerson Zaverucha
Systems Engineering and Computer Science, COPPE
Universidade Federal do Rio de Janeiro
Rio de Janeiro, RJ, Brazil
gerson@cos.ufrj.br

Artur d'Avila Garcez
Dept. of Computer Science
City St George's, University of London
London, UK
a.garcez@city.ac.uk

***Abstract*—Traditional neural networks (NNs) learn primarily from data, which limits their capacity to represent relational knowledge or handle symbolic relational data effectively. Although graph neural networks (GNNs) address this limitation at the level of relational data, they continue to struggle at learning relational knowledge. Neural-symbolic learning offers a solution by combining machine learning with knowledge representation, enabling the development of interpretable logic-based models learned from neural networks. Bottom clause propositionalization (BCP) is a prominent approach that transforms relational knowledge into attribute-value examples. A bottom clause is a logical representation created from each example as a starting point for the search process. BCP can be used with symbolic learners or neural networks to tackle relational domains. However, BCP often faces significant memory storage problems when handling larger datasets due to the volume of logical literals that it generates. Semi-propositionalization can alleviate these storage problems by grouping logical literals. However, it does not eliminate the substantial time requirements to create a bottom clause for each example. This paper investigates the application of sampling to the examples used for bottom clause generation. The hypothesis is that the number of examples needed to generate bottom clauses can be reduced significantly. Finding representative bottom clauses from data should enable relational learning to take place at an adequate level of abstract relational knowledge rather than simply at the level of the relations between any two data points. We evaluate this hypothesis by training a classifier with different sampling from five relational datasets. We experimentally validate the size of each sampling for each dataset. Experimental results show that training classifiers with fewer relational examples produces competitive results compared to using the entire dataset. The best results are obtained with up to 50% reduction in the set of examples.**

***Index Terms*—Relational Learning, Propositionalization, Neural-Symbolic Learning, Sampling**

I. Introduction

Traditional Neural Networks (NNs) for tabular data expect inputs to be in an attribute-value format (i.e. propositional representation). This type of representation limits such models from directly handling data composed of entities and their multiple interactions, often organized in relational format, e.g. *parent(X,Y)* denoting that a person X is a parent of a person Y for all the valid instances of the pair (X, Y). Graph Neural Networks (GNNs) have been deployed to learn from such structured data but have struggled at learning general relational knowledge [1]. Neural-symbolic systems [2] have emerged to overcome such limitations by combining machine learning with knowledge representation to enable the use of interpretable logic-based models with NNs. Logical and relational learning focus on learning first-order logic theories from examples [3], [4]. This allows for more expressive knowledge representation than propositional logic for learning, as it enables representing domains with multiple entities and their relationships using first-order logical formalisms. Logical learning also offers interpretable models with rich representations, allowing for learning and reasoning with multiple levels of abstraction [5], [6]. One example of a rule learned by this kind of model would be *daughter(X,Y)* $:-$ *parent(Y,X)*,*female(X)*. It states that X is a daughter of Y if Y is a parent of X and X is female. If X is not a daughter of Y, it is easy to see that it must be because Y is not a parent of X or X is not female. Learning from such representations directly from logical search usually requires long training times, as observed in most Inductive Logic Programming (ILP) systems [7]. One way to alleviate such a problem while handling relational data is to rely on Propositionalization [8].

Propositionalization focuses on transforming a relational task into an attribute-value learning task. It requires building features that capture relational properties of learning examples representing background knowledge. This enables complex relational learning tasks to be solved by propositional rule learning systems, i.e. traditional machine learning methods [9]. One such method is Bottom Clause Propositionalization (BCP) [10], which aims to transform a relational representation into a propositional one by constructing bottom clauses. A bottom clause (BC) is a conjunction of literals that covers an example in a hypothesis space concerning the background knowledge [11], [12]. The BCP algorithm focuses on creating one bottom clause for each training example so that no information is lost [10]. Then, the set of body

literals in the bottom clauses can be used as features in an attribute-value table for NN training, in which the columns are representations of such literals considering all examples, and each row represents an example.

Considering the bottom clauses generated by BCP, Semi-Propositionalization [8] techniques can be applied to group first-order literals to define propositional features. Grouping such literals reduces the size of feature vectors, which is usually a bottleneck when applying BCP due to the large number of features generated. Given the number of examples used to generate BCs, this process might cause significant memory storage challenges. This is a major problem when applying BCP to large datasets. Semi-Propositionalization for Bottom Clauses ($BCP_{semi-prop}$) [4] transforms each bottom clause into semi-propositional rules to create two or more rules that have equivalent meaning as the original first-order rule. Although $BCP_{semi-prop}$ overcomes the problem of generating very large vectors using BCP, creating one bottom clause for each example still requires a lot of time.

This paper uses sampling to select examples for bottom clause creation to reduce the training time. For larger datasets, not every data point is necessarily informative to guarantee better results [13]. We hypothesize that reducing the number of examples to be converted into a clause can produce a representative set of bottom clauses to generate suitable propositional values to be learned by a classifier. The size of each sample is empirically validated with our study concluding that some datasets require more examples than others to learn, with up to 50% decrease in the number of examples.

We evaluate our method using CILP++ [10], a Neural-Symbolic System for Relational Learning that combines BCP and another Neural-Symbolic System called $C - IL^2P$ [14], trained using samples of different sizes. We evaluate results on five relational datasets using seven different sampling rates for each dataset. We compare results using sampling with results using the entire training sets. Our experimental results show that sampling the training set has competitive results for most experiments compared to a model trained using all the training data. The best results are obtained with 50% of the data for most experiments, but some tasks require even less of the training set.

This paper is organized as follows. Section II introduces necessary background. Section III discusses related work on sampling-of-examples techniques in ILP. Section IV presents datasets used to validate this work and a short description of the system CILP++. Section V describes experimental results. Finally, Section VI presents conclusions and future work.

II. Key Concepts

We give an overview of concepts used to build the contributions of this paper, namely, Bottom Clause Propositionalization and Semi-Propositionalization for bottom clauses.

A. Bottom Clause Propositionalization

Propositionalization [9] is an ILP method that converts a relational task down to the propositional level by representing relational databases as attribute-value tables. It enables using traditional ML algorithms, such as decision trees and neural networks, which are fast attribute-value learners for inputs provided in a tabular format [4], [10].

Propositionalization algorithms use background knowledge and examples to build distinctive features to differentiate subsets of examples. To do this, such algorithms can be logic-oriented or database-oriented. Logic-oriented algorithms focus on building a set of relevant first-order features that distinguish between first-order objects. By contrast, database-oriented methods exploit database relations and functions to generate features for propositionalization [4], [10].

One example of a logic-oriented propositional approach is Bottom Clause Propositionalization (BCP) [10]. BCP searches for bottom clauses, which are elements proposed as part of the Progol system [15]. To give an overview of bottom clauses, we first present some definitions. One way to represent relational data is by using First-order Logic (FOL). FOL represents relations as logical facts formed by predicates and terms. Domains are represented using constants, variables, and predicates. Following Prolog [16] syntax, for example, *publication(title, janedoe)* is a logical fact that represents the relation between a published material identified by *title* and the person named as *janedoe* who wrote that material. In our example, *publication* is the predicate of this logical fact while *title* and *janedoe* are constants that represent objects in the real world. Terms can be variables, constants, or function symbols applied to terms. In Prolog, the names of variables start with capital letters, while the names of predicates and constants are lowercase. A literal can be an atom or a negated atom. Finally, a disjunction of literals forms a *clause*.

A bottom clause is the most specific clause within a hypothesis space that covers an example concerning background knowledge [12]. Bottom clauses are built from a single example using background knowledge and a language bias. Background knowledge is a set of logical facts or definite clauses, while language bias is a set of clauses that describe how these clauses can be built, how the search is done, and how far it can go. The language bias aims to make the bottom clause construction more efficient, as Progol searches for bottom clauses through a space of declarations that are modes for the hypothesized clause. The modes described in the language bias describe which predicates can appear in a clause's head or body. They also describe input and output variables, contacts, and an upper bound for how many times the specified predicates can appear in the same clause. Algorithm 1 offers a procedure that finds bottom clauses for a set of examples. As an example, consider the following background knowledge (BK) to learn the relation *motherInLaw* from one example of it, namely $motherInLaw(molly, harry)$:

$$BK = \begin{cases} mother(molly, ginny). \\ mother(lillian, harry). \\ wife(ginny, harry). \end{cases}$$

Progol would create a clause:

$$motherInLaw(A,B) :- mother(A,C), wife(C,B).$$

In [10], the authors propose modifying the BCP algorithm to allow the same hash function to be shared among all examples. This modification keeps variable associations consistent and allows negative examples to have bottom clauses (the original algorithm only deals with positive examples). By executing Algorithm 1 with $depth = 3$ on the positive example $motherInLaw(molly, harry)$ and on the negative example $motherInLaw(molly, lillian)$, it generates the following set:

$$E_{\perp} = \begin{cases} motherInLaw(A,B) : -mother(A,C), wife(C,B). \\ \neg\ motherInLaw(A,B) : -wife(A,C). \end{cases}$$

The literal *motherInLaw(A,B)* is called the *head* of the clause while predicates *mother(A,C)* and *wife(C,B)* compose its *body*. The algorithm uses determination predicates, prespecified relations that can appear in the clause's body. The building process is repeated until a number of cycles through the declarations is reached.

The next step of BCP is to convert each clause built above into an input vector. To do this, BCP uses the set of all body literals in the examples as possible features of a truth table to simplify a feature extraction process [10], [12]. In our example, this set is composed of three literals *mother(A,C), wife(C,B)* and *wife(A,C)*. Each literal is converted into a feature with an associated Boolean value to indicate whether it exists in the example. In our example, starting from the first bottom clause, a vector v_1 of size three is created. The first position corresponds to *mother(A,C)*, the second to *wife(C,B)*, and the third to *wife(A,C)*. The first two literals, i.e. *mother(A,C)* and *wife(C,B)*, appear in the first bottom clause, so the first two positions of v_1 receive value 1 ($v_1 = (1, 1, 0)$). For the second bottom clause, there is only *wife(A,C)*, thus $v_2 = (0, 0, 1)$. This step is presented in Algorithm 2.

Although BCP is a simple and fast-to-implement technique, it is a costly and time-consuming algorithm. Also, it can generate very large vectors of features, which makes generating bottom clauses for robust datasets prohibitive due to significant memory storage challenges [17]. To overcome the long time used to generate BCs, we proposed to use sampling. Addressing the significant memory storage problem can be done using Semi-Propositionalization, which we describe next.

B. Semi-Propositionalization

Semi-Propositionalization [8] reduces the size of vectors of features as it groups first-order literals to define a propositional feature. It uses semi-propositionalized rules, which cannot be decomposed into a set S of two or more rules with equivalent semantics to the original first-order rule [4].

Definition 1 (Semi-Propositional Rules [8]): Variables that occur at the head of the rule are called global variables. Variables that occur only in the body are called local variables. A Prolog rule in which there are no local variables is called

Algorithm 1: Bottom Clause Generation [10].

1 **Function** BC_GENERATION(E)**:**
2 $E_{\perp} \leftarrow \{\}$;
3 **for** $e \in E$ **do**
4 Add e to background knowledge and remove any previously inserted examples;
5 $inTerms \leftarrow \{\}$, $\perp_e \leftarrow \{\}$, $currDepth \leftarrow 0$;
6 Find the first *mode declaration* with head h which θ-subsumes e;
7 **for** $v/t \in \theta$ **do**
8 Replace v in h to t if v of type $\#$;
9 Replace v in h to v_k, where $k = hash(t)$, if v is of one of {+,-};
10 Add t to $inTerms$ if v of type $+$;
11 **end**
12 Add h to $\perp_e$;
13 **for** *each body mode declaration* b *and recall value* $recall$ **do**
14 **for** *substitutions* θ *of arguments* + *of* b *to elements of* $inTerms$ *and* $recall$ **do**
15 **while** $recall$ *number of iterations not reached* **do**
16 **if** *query(*$b\theta$*, backgroundKnowledge)* **then**
17 **for** $v/t \in \theta$ **do**
18 Replace v in b to t if v of type $\#$;
19 Replace v in b to v_k, where $k = hash(t)$ if v not of type $\#$;
20 Add t in $inTerms$ if v of type -;
21 **end**
22 Add $b\theta$ to $\perp_e$ if not added;
23 **end**
24 **end**
25 **end**
26 **end**
27 Increment $currDepth$;
28 Go back to line 13 if $currDepth < depth$;
29 Add negation symbol to the head of $\perp_e$ if e is a negative example;
30 Add $\perp_e$ to $E_{\perp}$;
31 **end**
32 **return** $E_{\perp}$

constrained. A constrained rule in which every global variable occurs once in every literal is called **semi-propositional**.

Definition 2 (First-Order Features [8]): For any two body literals L_1 and L_2 of a given rule, L_1 and L_2 belong to the same equivalence class $L_1 \sim_{lv} L_2$ iff they share a local variable. $\sim_{lv}$ is reflexive and asymmetric, and hence its transitive closure $=_{lv}$ is an equivalence relation inducing a partition on the body. The conjunction of literals in an equivalence class is called a **first-order feature**.

Proposition 1 (Decomposition of Rules [8]): Let R be a Prolog rule; R' is to be constructed as follows. Replace each first-order feature F in the body of R with a literal L consisting of a new predicate with $R's$ global variable(s) as argument(s), and add a rule $L :- F$. R' together with the newly constructed rules is equivalent to R because they have the same success set (the set of queries covered by a rule). R' is also a semi-propositional rule.

Algorithm 2: Attribute-value table generation [10].

```
Function TABLE_GENERATION(nLiterals, E⊥):
    Ev ← {};
    for ⊥e ∈ E⊥ do
        Create numerical vector vi of the size of nLiterals;
        Fill positions of vi with 0;
        Changes its values to 1 for each position corresponding to a body literal of ⊥e;
        Add vi to Ev;
        Associate a label 1 to vi in case of a positive example, and -1 otherwise;
    end
return Ev
```

This paper follows the method proposed by [4] called $BCP_{semi-prop}$. $BCP_{semi-prop}$ aims at transforming each bottom clause into semi-propositional rules. The features with a first-order description are obtained by applying Proposition 1. To illustrate the method proposed by [4], consider the following example, which contains a single bottom clause $R_\perp$:

$$motherInLaw(A,B) :- parent(A,C), wife(C,B), wife(A,D), brother(B,D).$$

BCP considers each literal as a feature itself, so, in this case, it would output four features: *parent(A,C),wife(C,B),wife(A,D),brother(B,D)*.

Following Definition 1, the head of the rule, i.e. *motherInLaw(A,B)*, contains two global variables A and B. Variables C and D are local variables as they only appear in literals present in the body of the rule. Then, Definition 2 states that literals belong to the same equivalence class iff they share a local variable. The algorithm iterates through the set of body literals to create a set of first-order features. Starting from *parent(A,C)*, we can notice that it shares a local variable (C) with another literal, i.e. *wife(C,B)*. Since A and B are global variables, it creates a first-order feature *parent(A,C), wife(C,B)*. The same happens for *wife(A,D)* and *brother(B,D)* as they share a local variable D. Finally, the set of first-order features for $R_\perp$ are:

$$F_1 = \{parent(A,C), wife(C,B)\}$$
$$F_2 = \{wife(A,D), brother(B,D)\}$$

$BCP_{semi-prop}$ treats each decomposition as a feature and generates two clauses from the decomposition of $R_\perp$:

$$L_1(A,B) :- parent(A,C), wife(C,B)$$
$$L_2(A,B) :- wife(A,D), brother(B,D)$$

In the end, we can rewrite $R_\perp$ as the following semi-propositional rule $R'_\perp$:

$$motherInLaw(A,B) :- L_1(A,B), L_2(A,B)$$

This reduces the size of the vector of features from four to two, which helps avoid memory storage problems.

III. Related Work

The time taken by ILP approaches is proportional to the number of examples [13]. This is due to the theorem-proving effort required to evaluate the built theory. In some cases, it is not feasible for an ILP algorithm to use the entire dataset in a single step. To handle such cases, [13] investigate two methods: (i) subsampling of data and (ii) logical windowing to construct theories by sampling small fractions of data. The authors investigate the estimated predictive accuracy and the time required to create the theory with and without sampling techniques using CProgol. Empirical results have shown that ILP systems that use a sampling scheme can build an accurate theory faster than systems that use all available data [13]. Similarly, [18] also describe that ILP approaches have poor performance for large datasets (or data streams [19]) and perform a comparison analysis between proposed ILP systems.

Sampling is widely used in ILP methods to reduce the complexity of computing scores [13], [18]. One example is CART [20], a propositional learner that uses subsampling to reduce sample size. To do it, an upper limit is given on the number of examples used to evaluate the best attribute to select nodes when building trees. Examples are randomly selected, and the attribute that best discriminates between classes in this sample is the one selected. Then, the tree construction continues using all examples (no sampling is performed) onto the next level to avoid being left with fewer cases as the depth of the tree increases. All classes must be well represented in the selected random sample. In [13], authors consider the sampling used by CART, in which the sample size is determined considering the number of positive and negative examples, and the data distributions.

Logical windowing focuses on incrementally building theories using sampling. It is a method developed for decision tree-based approaches to deal with large datasets [13]. Examples of

such methods are ID3 and C4.5. In this case, a small sample (window) of fixed size is selected from the examples during training. Then, the tree is evaluated considering all examples. Suppose the number of examples classified incorrectly is unacceptable. In that case, a small sample of the incorrectly classified examples is added to the training set, and a new tree is built. This process is repeated until the number of incorrectly classified examples is acceptable. For the subsampling method, the window size is determined considering the number of examples and the number of positive and negative examples. Finally, sampling of examples is also applied in [21] to speed up training a large dataset using Markov logic networks using gradient-based boosting. In the experiments, the authors choose to use 25% examples of a large dataset to reduce training time. In [22], authors also drastically subsample the number of examples of one of the datasets used in experiments to learn the same SRL model.

In this work, we propose applying sampling to BCP_{semi_prop} to reduce the effort to transform relational data into attribute-value tables. Our method is more straightforward than ones mentioned above, as we propose randomly selecting sample examples. We tested seven different samples sizes to evaluate the accuracy and the time needed to generate clauses.

IV. Materials and Methods

a) Datasets: To evaluate the proposed method, we use five relational datasets commonly used in the previous literature [23]–[26]: IMDB, Cora, UWCSE, Yeast, and NELL. (1) The IMDB data set [27] has the objective of learning the relation *workedunder* that describes whether an actor has worked for a director. Its background knowledge contains information about movies, such as *actor*, *director*, and *genre*. It is divided into five mega-examples that contain information about four movies. (2) Cora [28] contains information about Computer Science papers like their *title*, *author*, and *venue*. It consists of 1295 citations to 122 papers to predict if two venues represent the same conference. It is also divided into five mega-examples. (3) UWCSE [29] contains information about the Department of Computer Science and Engineering at the University of Washington (UWCSE) since the objective is to learn the relation *advisedby*, i.e., if a student is advised by a professor, using information about publications, their authors, project and members, course levels, and more. It is divided into five mega-examples. (4) Yeast protein [30] is a dataset obtained from MIPS[1] Comprehensive Yeast Genome Database. It provides information about proteins, such as their location, function, enzyme, complex, and phenotype, to learn if a protein is associated with a class. It is divided into four folds, independent of each other. (5) NELL [31] is a machine learning system that extracts information from web texts and converts it into a probabilistic knowledge base. We consider two domains from NELL: Sports and Finances. NELL Sports contains information such as athletes and their teams, leagues and their teams, etc, as we want to predict which sport is played by a team. On the other hand, NELL Finances contains information about economic sectors of companies, companies' CEOs, companies' country, etc, to predict if a company belongs to an economic sector. We split the two datasets into three different folds following [23]. Statistics about the datasets are presented in Table I.

b) CILP++: To evaluate our proposed method, we analyze the performance of CILP++ considering different sizes of sampling from the training set. CILP++ consists of a BCP and NN components to learn the attribute-value tables generated by BCP. We implemented the NN component using a MLPClassifier [2]. In this case, the number of input neurons corresponds to the number of features generated by BCP_{semi_prop}, the activation function is the hyperbolic tan function ($f(x) = tanh(x)$), and we set Adam as the optimizer, which is based on stochastic gradient descent. Finally, we use a single hidden layer with 100 neurons, which leads to the best results after empirical evaluation.

V. Experiments and Results

In this section, we present the experiments performed to evaluate the proposed method and the results obtained from them. We conducted a set of experiments to investigate the following research questions.

Q1 Does bottom clause sampling have competitive results compared to results using all available data?

Q2 Is there a universally good sampling rate for all datasets?

We evaluate the classifier's performance for each dataset, considering different sampling rates. To do it, we use one fold for training, one for validation, and one for testing. We test five sampling rates: 1%, 5%, 10%, 20%, 30%, 40%, and 50%. Training data are sampled at random for each sampling rate. We also compare results when using all examples for training. To compare the performance for different rates, we used the area under the ROC curve (AUC ROC), and the area under the PR curve (AUC PR) [32]. We also present the time needed to build bottom clauses. We do not consider the time required to train the model and generate the respective feature vectors because they are negligible compared to the time needed to create bottom clauses. As the size of such vectors of features can be a bottleneck when using BCP, we implement BCP_{semi_prop}, and present the number of features generated for each sample size. Considering all bottom clauses generated for each sampling rate, this number is the maximum number of literals. We do not obtain rules from the trained NN, but how to obtain such rules is presented in [5].

Tables II, III, IV, V, VI, and VII present the experimental results for IMDB, UWCSE, Cora, Yeast, NELL Sports, and NELL Finances, respectively. We present the average of five runs, and the best results are in bold. Sampling is done in a ratio of two negatives for one positive. We also present the results using the training set to validate whether the model learns properly from the data used during the training. Our

[1] Munich Information Center of Protein Sequence

[2] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

experimental results are competitive when sampling is used for most experiments. The exception is Yeast, whose results using sampling are not competitive compared to results using the entire training set.

The experimental results using IMDB show that sampling training examples does not affect performance, and that the model has its best results when using 50% of the training set. However, we note that competitive results are obtained using fewer samples like 10%, 20%, and 30% of the data. The second-best results are obtained when sampling 40% of the data. Despite the small difference between training times, results show that training time does grow with the number of examples, like the number of features generated, and by using sampling, we can reduce the time to generate the bottom clause by more than 70% without impairing performance. The same results can be observed for UWCSE. Table III shows competitive results when the sample size is equal to or greater than 30%. For AUC PR, the best performance is reached when using 50% of the training examples. When training using Cora, there are no competitive results for AUC ROC, as the models using sampling underperform the model trained using all available data. The best performance is reached when all the data is used. When using sampling, the best results are obtained when using 5% of data for AUC ROC and 1% for AUC PR. As shown in Table IV, the model overfits for both AUC ROC and AUC PR when the proportion of training data increases. The results for Yeast show that the model benefits from more examples during the learning. The best performance is obtained when using 40% of the data, but it is far from the model's best performance, obtained when using the entire training set. This might indicate Cora and Yeast are complex and hard-to-learn datasets, while IMDB and UWCSE can be learned with fewer examples. When using NELL Sports, Table VI shows that sampling results are competitive. In this case, the best performance is obtained using 40% of data for both AUC ROC and AUC PR. The second-best results are obtained when using 50% of the data. Finally, for NELL Finances, Table VII shows competitive results when using 20% of data and more. In addition, the model trained using 1% of the data outperforms other models for AUC PR. Observing the results when using the training set to verify the model's efficiency when learning NELL Finances, we can observe some difficulty even when using all the available data. Nevertheless, validation results are very close to using all data when the algorithm is trained using 50% of the training data.

The number of features for all experiments grows with the number of examples. The time to generate BCs is also proportional to the number of examples used during training. Although the difference between the time needed to generate BCs using sampling and using all examples seems insignificant for smaller datasets, sampling makes a bigger difference in time-saving for large datasets such as Cora, Yeast, and NELL. In addition to being the largest datasets, they also present the largest bottom clauses in number of literals as presented in Tables IV, V, and VI. Finally, Table VIII shows the results of test sets considering the best and second sampling rates that give the best results for validation. The best results are in bold. We measured the statistical significance between different sampling rates and the model trained using the entire training set using a paired t-test with $p \leq 0.05$. In Table VIII, $\star$ indicates no statistical difference between the results obtained through sampling and those using the entire training set. We can observe that performance decreases compared to results using the validation sets, which is expected. For IMDB, UWCSE, and Cora, results are competitive compared to when no sampling is applied. In this case, results for Cora with no sampling are similar to those reported by [23], [24] when applying a transfer learning-based approach. We also have competitive results for AUC PR when using NELL. But sampling impairs performance for Yeast and AUC ROC for NELL Sports. Finally, we can answer Q1 positively, as sampling does not impair performance for most experiments. But, there is no universal sampling rate for all datasets as some datasets need more data during learning than others (Q2).

TABLE I: Statistics of the datasets used to evaluate our proposal.

Dataset	No. of Constants	No. of Types	No. of Predicates	No. of Positive Examples	Total No. of ground literals
IMDB	297	3	6	382	71824
UWCSE	914	9	14	113	16900
Cora	2457	5	10	3017	152100
Yeast	2470	7	7	369	40128
NELL Sports	4538	4	8	397	4323
NELL Finances	3340	5	10	778	51578

VI. Conclusion

This paper investigated the use of sampling for bottom clause propositionalization. The hypothesis is that obtaining a representative set of bottom clauses to generate propositional values is possible by reducing the number of examples used to train a classifier. Applying sampling can avoid memory storage problems, which can be a major problem when applying BCP to more robust datasets due to the high number of literals. To evaluate our approach, we use CILP++, a Neural-Symbolic System for Relational Learning that combines BCP_{semi_prop} and a Neural-Symbolic System called $C - IL^2P$, trained using different sizes of training data on five datasets. We also compare results with the same machine learning model when trained using the entire dataset. Experimental results show that reducing the size of training examples offers competitive results compared to the results when no sampling is applied. In this way, applying BCP_{semi_prop} with less training time without impairing performance is possible. Our results also show that no common sampling rate fits all datasets, as some need more data than others. As future work, we envision that sampling examples for BCP can benefit transfer learning-based approaches, such as TreeBoostler [23] and TransBoostler [24]–[26]. Further, as data is sampled randomly, we envision that different sampling methods can be applied to improve our method and help build a more representative training set.

TABLE II: Training and validation results for AUC ROC and AUC PR when using IMDB considering different sampling rates. We also present the time needed to generate BCs and the size of attribute-value tables.

		1%	5%	10%	20%	30%	40%	50%	100%
Training	AUC ROC	0.450	0.700	1.000	1.000	1.000	1.000	1.000	1.000
	AUC PR	0.566	0.800	1.000	1.000	1.000	1.000	1.000	1.000
Validation	AUC ROC	0.431	0.636	0.863	0.872	0.901	0.913	**0.934**	0.934
	AUC PR	0.573	0.717	0.825	0.836	0.861	0.874	**0.902**	0.896
Time to generate BCs (s)		0.40	0.41	0.43	0.47	0.50	0.55	0.58	0.76
Number of features for attribute-value tables		6	13	11	17	25	23	27	47

TABLE III: Training and validation results for AUC ROC and AUC PR when using UWCSE considering different sampling rates. We also present the time needed to generate BCs and the size of attribute-value tables.

		1%	5%	10%	20%	30%	40%	50%	100%
Training	AUC ROC	0.900	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	AUC PR	0.950	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Validation	AUC ROC	0.482	0.447	0.530	0.717	0.840	0.905	**0.910**	0.732
	AUC PR	0.578	0.558	0.575	0.693	0.821	0.863	**0.868**	0.741
Time to generate BCs (s)		0.46	0.50	0.56	0.80	0.91	1.07	1.20	1.95
Number of features for attribute-value tables		19	90	119	201	248	273	350	555

TABLE IV: Training and validation results for AUC ROC and AUC PR when using Cora considering different sampling rates. We also present the time needed to generate BCs and the size of attribute-value tables.

		1%	5%	10%	20%	30%	40%	50%	100%
Training	AUC ROC	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	AUC PR	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Validation	AUC ROC	0.515	**0.539**	0.516	0.458	0.428	0.393	0.404	**0.714**
	AUC PR	**0.779**	0.689	0.714	0.670	0.656	0.651	0.650	**0.830**
Time to generate BCs (s)		0.94	38.50	42.29	95.71	125.05	186.49	227.77	449.43
Number of features for attribute-value tables		74	332	636	1132	1559	1983	2411	4136

TABLE V: Training and validation results for AUC ROC and AUC PR when using Yeast considering different sampling rates. We also present the time needed to generate BCs and the size of attribute-value tables.

		1%	5%	10%	20%	30%	40%	50%	100%
Training	AUC ROC	1.000	1.000	0.977	0.983	0.988	0.985	0.986	0.988
	AUC PR	1.000	1.000	0.985	0.988	0.992	0.990	0.990	0.992
Validation	AUC ROC	0.273	0.384	0.384	0.475	0.460	**0.547**	0.482	**0.872**
	AUC PR	0.376	0.560	0.561	0.644	0.631	**0.675**	0.651	**0.831**
Time to generate BCs (s)		0.83	1.40	2.44	4.86	7.65	9.55	11.98	22.81
Number of features for attribute-value tables		22	51	73	167	187	311	394	636

TABLE VI: Training and validation results for AUC ROC and AUC PR when using NELL Sports considering different sampling rates. We also present the time needed to generate BCs and the size of attribute-value tables.

		1%	5%	10%	20%	30%	40%	50%	100%
Training	AUC ROC	0.600	0.785	0.785	0.796	0.800	0.796	0.798	0.800
	AUC PR	0.933	0.912	0.922	0.919	0.921	0.923	0.921	0.921
Validation	AUC ROC	0.440	0.609	0.649	0.703	0.725	**0.729**	0.728	**0.808**
	AUC PR	0.817	0.830	0.842	0.852	0.857	**0.858**	0.857	**0.919**
Time to generate BCs (s)		96.88	365.29	804.14	1877.80	2114.32	2212.58	3855.08	6739.59
Number of features for attribute-value tables		136	1028	1461	1724	2429	2728	3023	4731

TABLE VII: Training and validation results for AUC ROC and AUC PR when using NELL Finances considering different sampling rates. We also present the time needed to generate BCs and the size of attribute-value tables.

		1%	5%	10%	20%	30%	40%	50%	100%
Training	AUC ROC	0.500	0.615	0.673	0.702	0.628	0.625	0.615	0.603
	AUC PR	0.800	0.948	0.880	0.883	0.880	0.857	0.885	0.857
Validation	AUC ROC	0.500	0.485	0.497	0.533	0.545	0.528	**0.548**	**0.557**
	AUC PR	**0.853**	0.700	0.723	0.783	0.791	0.776	0.797	0.803
Time to generate BCs (s)		0.59	0.85	0.96	1.52	2.07	2.26	2.72	4.90
Number of features for attribute-value tables		2	72	95	140	145	192	243	323

TABLE VIII: Results for test sets using the sampling rates that give best and second-best results for validation sets compared to models trained using all available data. ⋆ indicates no statistical difference between results with and without sampling.

	Best result (rate)		Second best result (rate)		Result using all training set	
	AUC ROC	AUC PR	AUC ROC	AUC PR	AUC ROC	AUC PR
IMDB	0.807⋆ (50%)	**0.794**⋆ (50%)	0.776⋆ (40%)	0.749⋆ (40%)	**0.822**	0.791
UWCSE	**0.483**⋆ (50%)	**0.625**⋆ (50%)	0.388⋆ (40%)	0.540⋆ (40%)	0.394	0.566
Cora	0.538 (5%)	0.566 (1%)	0.585 (10%)	0.542 (10%)	**0.649**	**0.644**
Yeast	0.555 (40%)	0.682 (40%)	0.467 (50%)	0.637 (50%)	**0.841**	**0.803**
NELL Sports	0.675⋆ (40%)	0.817⋆ (40%)	0.666 (50%)	0.790⋆ (30%)	**0.814**	**0.906**
NELL Finances	0.565 (50%)	0.788 (50%)	0.545 (30%)	0.791 (30%)	**0.803**	**0.814**

ACKNOWLEDGMENT

This research was partially financed by CNPq (National Council for Scientific and Technological Development) and FAPERJ - *Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro*.

REFERENCES

[1] H. Ren and J. Leskovec, "Beta embeddings for multi-hop logical reasoning in knowledge graphs," in *Adv. in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 19716–19726, Curran Associates, Inc., 2020.

[2] T. R. Besold, A. S. d'Avila Garcez, S. Bader, H. Bowman, P. M. Domingos, P. Hitzler, K. Kühnberger, L. C. Lamb, P. M. V. Lima, L. de Penning, G. Pinkas, H. Poon, and G. Zaverucha, "Neural-symbolic learning and reasoning: A survey and interpretation," in *Neuro-Symbolic Artificial Intelligence: The State of the Art* (P. Hitzler and M. K. Sarker, eds.), vol. 342 of *Frontiers in Artificial Intelligence and Applications*, pp. 1–51, IOS Press, 2021.

[3] L. D. Raedt, *Logical and relational learning*. Cognitive Technologies, Springer, 2008.

[4] M. V. M. França, G. Zaverucha, and A. S. d. Garcez, "Neural relational learning through semi-propositionalization of bottom clauses," in *2015 AAAI Spring Symposium Series*, 2015.

[5] M. V. M. França, A. S. d'Avila Garcez, and G. Zaverucha, "Relational knowledge extraction from neural networks," in *Proc. of the NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches co-located with the 29th Annual Conference on Neural Information Processing Systems (NIPS 2015), Montreal, Canada, December 11-12, 2015* (T. R. Besold, A. S. d'Avila Garcez, G. F. Marcus, and R. Miikkulainen, eds.), vol. 1583 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015.

[6] N. Kaur, G. Kunapuli, S. Joshi, K. Kersting, and S. Natarajan, "Neural networks for relational data," in *Inductive Logic Programming: 29th Int. Conf., ILP 2019, Plovdiv, Bulgaria, September 3–5, 2019, Proceedings 29*, pp. 62–71, Springer, 2020.

[7] S. Muggleton and L. de Raedt, "Inductive logic programming: Theory and methods," *The J. of Logic Programming*, vol. 19-20, pp. 629–679, 1994. Special Issue: Ten Years of Logic Programming.

[8] N. Lavrač and P. A. Flach, "An extended transformation approach to inductive logic programming," *ACM Trans. Comput. Logic*, vol. 2, p. 458–494, oct 2001.

[9] S. Kramer, N. Lavrač, and P. Flach, *Propositionalization Approaches to Relational Data Mining*, pp. 262–291. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[10] M. V. França, G. Zaverucha, and A. S. d'Avila Garcez, "Fast relational learning using bottom clause propositionalization with artificial neural networks," *Mach. learn.*, vol. 94, pp. 81–104, 2014.

[11] S. Muggleton, "Inverse entailment and progol," *New generation computing*, vol. 13, pp. 245–286, 1995.

[12] A. Tamaddoni-Nezhad and S. Muggleton, "The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause," *Mach. learn.*, vol. 76, pp. 37–72, 2009.

[13] A. Srinivasan, "A study of two sampling methods for analyzing large datasets with ilp," *Data Mining and Knowledge Discovery*, vol. 3, pp. 95–123, 1999.

[14] A. S. Avila Garcez and G. Zaverucha, "The connectionist inductive learning and logic programming system," *Applied Intelligence*, vol. 11, pp. 59–77, 1999.

[15] S. H. Muggleton, "Inverse entailment and progol," *New Gener. Comput.*, vol. 13, no. 3&4, pp. 245–286, 1995.

[16] I. Bratko, *PROLOG Programming for Artificial Intelligence*. USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 1990.

[17] T. Luca, A. Paes, and G. Zaverucha, "Select first, transfer later: Choosing proper datasets for statistical relational transfer learning," in *Int. Conf. on Inductive Logic Programming*, pp. 62–76, Springer, 2023.

[18] G. Zaverucha and P. Cardoso Motta, "Comparative evaluation of approaches to scale up ilp," in *Inductive Logic Programming: 16th Int. Conf. on Inductive Logic Programming, ILP 2006, Santiago de Compostela, Spain, 2006, Short Papers of the 16th Int. Conf. on Inductive Logic Programming (ILP 2006)*, pp. 37–39, UDC Press, 2006.

[19] C. Lopes and G. Zaverucha, "Htilde: scaling up relational decision trees for very large databases," in *Proc. of the 2009 ACM Symposium on Applied Computing*, SAC '09, (New York, NY, USA), p. 1475–1479, Association for Computing Machinery, 2009.

[20] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Cart," *Classification and regression trees*, 1984.

[21] T. Khot, S. Natarajan, K. Kersting, and J. Shavlik, "Learning markov logic networks via functional gradient boosting," in *2011 IEEE 11th int. conf. on data mining*, pp. 320–329, IEEE, 2011.

[22] J. Davis, I. M. Ong, J. Struyf, E. S. Burnside, D. Page, and V. S. Costa, "Change of representation for statistical relational learning.," in *IJCAI*, pp. 2719–2726, 2007.

[23] R. Azevedo Santos, A. Paes, and G. Zaverucha, "Transfer learning by mapping and revising boosted relational dependency networks," *Mach. Learn.*, vol. 109, pp. 1435–1463, 2020.

[24] T. Luca, A. Paes, and G. Zaverucha, "Mapping across relational domains for transfer learning with word embeddings-based similarity," in *Int. Conf. on Inductive Logic Programming*, pp. 167–182, Springer, 2021.

[25] T. Luca, A. Paes, and G. Zaverucha, "Combining word embeddings-based similarity measures for transfer learning across relational domains," in *Inductive Logic Programming* (S. H. Muggleton and A. Tamaddoni-Nezhad, eds.), (Cham), pp. 84–99, Springer Nature Switzerland, 2024.

[26] T. Luca, A. Paes, and G. Zaverucha, "Word embeddings-based transfer learning for boosted relational dependency networks," *Mach. Learn.*, vol. 113, no. 3, pp. 1269–1302, 2024.

[27] L. Mihalkova and R. J. Mooney, "Bottom-up learning of markov logic network structure," in *Proc. of the 24th Int. Conf. on Mach. Learn.*, ICML '07, (New York, NY, USA), p. 625–632, Association for Computing Machinery, 2007.

[28] M. Bilenko and R. J. Mooney, "Adaptive duplicate detection using learnable string similarity measures," in *Proc. of the Ninth ACM SIGKDD Int. Conf on Knowledge Discovery and Data Mining*, KDD '03, (New York, NY, USA), p. 39–48, Association for Computing Machinery, 2003.

[29] H. Khosravi, O. Schulte, J. Hu, and T. Gao, "Learning compact markov logic networks with decision trees," *Mach. learn.*, vol. 89, no. 3, pp. 257–277, 2012.

[30] H.-W. Mewes, D. Frishman, U. Güldener, G. Mannhaupt, K. Mayer, M. Mokrejs, B. Morgenstern, M. Münsterkötter, S. Rudd, and B. Weil, "Mips: a database for genomes and protein sequences," *Nucleic acids research*, vol. 30, no. 1, pp. 31–34, 2002.

[31] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, and T. M. Mitchell, "Toward an architecture for never-ending language learning," in *Proc. of the Twenty-Fourth AAAI Conf. on Artificial Intelligence*, AAAI'10, p. 1306–1313, AAAI Press, 2010.

[32] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proc. of the 23rd Int. Conf. on Machine learning*, pp. 233–240, 2006.