Principled and Efficient Motif Finding for Structure Learning of Lifted Graphical Models

Jonathan Feldstein[*1,2], **Dominic Phillips**[*1], **Efthymia Tsamoura**[3]
[1] University of Edinburgh, Edinburgh, United Kingdom
[2] BENNU.AI, Edinburgh, United Kingdom
[3] Samsung AI, Cambridge, United Kingdom
jonathan.feldstein@bennu.ai, dominic.phillips@ed.ac.uk, efi.tsamoura@samsung.com

Abstract

Structure learning is a core problem in AI central to the fields of *neuro-symbolic AI* and *statistical relational learning*. It consists in automatically learning a logical theory from data. The basis for structure learning is mining repeating patterns in the data, known as *structural motifs*. Finding these patterns reduces the exponential search space and therefore guides the learning of formulas. Despite the importance of motif learning, it is still not well understood. We present the first principled approach for mining structural motifs in *lifted graphical models*, languages that blend first-order logic with probabilistic models, which uses a stochastic process to measure the similarity of entities in the data.

Our first contribution is an algorithm, which depends on two intuitive hyperparameters: one controlling the uncertainty in the entity similarity measure, and one controlling the softness of the resulting rules. Our second contribution is a preprocessing step where we perform hierarchical clustering on the data to reduce the search space to the most relevant data. Our third contribution is to introduce an $\mathcal{O}(n \ln n)$ (in the size of the entities in the data) algorithm for clustering structurally-related data. We evaluate our approach using standard benchmarks and show that we outperform state-of-the-art structure learning approaches by up to 6% in terms of accuracy and up to 80% in terms of runtime.

1 Introduction

Motivation In artificial intelligence, combining statistical and logical representations is a long-standing and challenging aim. The motivation behind combining the two is that logical models can represent heterogenous data and capture causality, while statistical models handle uncertainty (Getoor and Taskar 2007; Russell 2015). General approaches to represent structural information are *lifted graphical models* (LGMs), such as *Markov logic networks* (MLNs) (Richardson and Domingos 2006) and *probabilistic soft logic* (PSL) (Bach et al. 2017). These are languages that define Markov random fields in a declarative fashion and are represented as theories of weighted formulas in first-order logic. The versatility of LGMs is reflected in their variety of applications, including bioinformatics (Lippi and Frasconi 2009), natural language understanding(Wu and Weld 2008), entity linking (Singla and Domingos 2006) and others (Chen and Wang 2014; Ha et al. 2011; Riedel and Meza-Ruiz 2008; Crane and Mcdowell 2012). Recently, they have also been adopted in neurosymbolic frameworks (Wang and Poon 2018; Hu et al. 2016).

Unsurprisingly, the quality of a logical theory, that is, the extent to which it models the task it is supposed to solve, has a strong impact on the performance of the downstream applications. Manually optimising formulae to boost performance is a costly, time-consuming and error-prone process that restricts the scope of application. This can raise fundamental criticism against frameworks that require such theories as part of their input (Manhaeve et al. 2018; Yang, Ishay, and Lee 2020). An alternative is the automated learning of LGMs from data, a problem known as *structure learning*. The ultimate goal is to design a general framework that can efficiently learn high-quality models on large datasets in a principled fashion. Several pioneering structure learning algorithms have been developed for MLNs (Kok and Domingos 2005; Mihalkova and Mooney 2007; Kok and Domingos 2010).

Problem Generally, structure learning consists in searching for formulae in an exponential search space. The naive approach would consist in trying every possible combination of predicates and logical connectives, which is computationally expensive (Kok and Domingos 2005). Therefore, to reduce computational complexity, every sophisticated structure learner proceeds by searching for formulae within templates. These templates can be user-defined or learnt automatically (Mihalkova and Mooney 2007; Kok et al. 2007). Every sophisticated learner can thus be summarized in three main steps: S1 - Apply heuristics to abstract-out common, recurrent patterns within the data to be used as templates. S2 - Iteratively generate formulae based on the previously found patterns and evaluate candidate formulae based on how well they generalize to the training data. S3 - Learn the collective weights of the optimal formulae. Remark that finding good templates is the basis for successful structural learning, as it not only reduces the search space but also forms the starting point of the structure learning algorithm and constrains the shape of logical formulae generated in later stages.

For example, the state-of-the-art structure learning algorithm, *Learning using Structural Motifs* (LSM), reduces

[*]These authors contributed equally.

the search space for formulae by focusing within recurring patterns of commonly connected entities in the relational database (Kok and Domingos 2010). The task of mining these patterns involves repeatedly partitioning the entities of a database into symmetrically-equivalent sets relative to a reference entity. These sets are called *(structural) motifs*. Since the entities in a structural motif are symmetric, formula learning only needs to be performed on one entity instead of each separately. Therefore, structural motifs guide the groundings of potential logical formulas of the LGM (S1).

The key difference between structure learners that do not require user input is how the templates are found. Still, the state-of-the-art suffers from several shortcomings that have a negative impact on the scalability and effectiveness of the full pipeline (S1-S3). Firstly, the symmetry-partitioning algorithm has six unintuitive hyperparameters that need to be calibrated to each dataset. The difficulty of finding these parameters can lead to inaccurate partitioning. Secondly, the main clustering algorithm, a core step to obtain symmetric partitions, has $\mathcal{O}(n^3)$ complexity in the number of entities to partition. This can result in significant slowdowns on databases that are densely connected.

Contributions In this work, we design a more principled and scalable algorithm for extracting motifs through symmetry-partitioning (stage S1 of structure learning). In our algorithm, we make three key contributions that overcome the limitations of prior art. In Section 3, we address the first limitation and propose a *principled* algorithm using the theoretic properties of hypergraphs to design an approach that uses just two, intuitive hyperparameters: one that controls the uncertainty of the similarity measure of entities in the data and one that controls the softness of the resulting formulae. In Section 4, we tackle the issue of *efficiency*. Firstly, we design an alternative $\mathcal{O}(n \ln n)$ symmetry-partitioning algorithm. Secondly, we propose a pre-processing step where we hierarchically cluster the relational database to reduce the required computation and further improve the guiding of the formulae finding. Beyond the above contributions, we present PRISM (PRincipled Identification of Structural Motifs) a parallelized, flexible, and optimized C++ implementation of the entire algorithm[1]. In Section 6, we assess the performance of the developed techniques against LSM and BOOSTR on datasets used as standard benchmarks in the literature.

2 Preliminaries

A ***hypergraph*** $\mathcal{H} = (V, E)$ is a pair of sets of nodes $V = \{v_i\}_{i=0}^{|V|}$ and hyperedges $E = \{e_i\}_{i=0}^{|E|}$. A hyperedge $e_i \in E$ is a non-empty subset of the nodes in $\mathcal{H}$. A hypergraph $\mathcal{H}$ is labelled, if each hyperedge in $\mathcal{H}$ is labelled with a categorical value. We use $\mathsf{label}(e_i)$ to denote the label of the hyperedge e_i. A ***path*** π of length L in $\mathcal{H}$ is an alternating sequence of nodes v_i and hyperedges e_i, such that $v_i, v_{i+1} \in e_i$, of the form $(v_0, e_0, v_1, \ldots, v_{L-1}, e_{L-1}, v_L)$ for $0 \leq i \leq L-1$. The ***diameter*** of $\mathcal{H}$ is the maximum length of the shortest path between any two nodes v_i, v_j in $\mathcal{H}$. The ***signature*** of a path π is the sequence of the labels of the edges occurring in π, i.e., $(\mathsf{label}(e_0), \ldots, \mathsf{label}(e_{L-1}))$.

A ***relational database*** $\mathcal{D}$ can be represented by a hypergraph $\mathcal{H} = (V, E)$ by defining V to be the union of the constants in $\mathcal{D}$, and defining E such that every k-ary ground atom $R(c_1, \ldots, c_k)$ in $\mathcal{D}$ becomes a hyperedge $e \in E$, with label R, whose elements are the nodes corresponding to the constants $c_1, \ldots, c_n$.

A ***random walk*** on $\mathcal{H}$ is a stochastic process that generates paths by traversing edges in $\mathcal{H}$. The ***length*** of a random walk is defined as the number of edges traversed in the path. Let v_i and v_j be two nodes in $\mathcal{H}$. The ***hitting time*** $h_{i,j}$ from v_i to v_j is the average number of steps required to reach v_j for the first time with random walks starting from v_i.

The ***L-truncated hitting time*** h_{ij}^L (THT) is the hitting time where the length of the random walk is limited to at most L steps. It is defined recursively as $h_{ij}^L = 1 + \sum_k p_{ik} h_{kj}^{L-1}$, where p_{ij} is the transition matrix of the random walk, with $h_{ij}^L = 0$ if $i = j$, and $h_{ij}^L = L$ if j is not reached in L steps. The more short paths that exist between v_i and v_j, the shorter the THT. The THT is therefore a measure of the connectedness of nodes.

We denote by $\mathcal{S}_{i,j}^L$ the set of path signatures of lengths up to L that start at v_i and end at v_j. The ***L-path signature distribution*** $P_{i,j}^L$ is then the probability distribution over the elements of $\mathcal{S}_{i,j}^L$ under a given random walk process. The ***marginal L-path signature distribution*** $P_{i,j}^L|_l$ is the marginal probability distribution when only paths of length *exactly* $l \in \{1, 2, \ldots, L\}$ are considered. The quantities $P_{i,j}^L(\sigma)$ and $P_{i,j}^L|_l(\sigma)$ respectively denote the probability and marginal probability of path signature σ. With this, we now introduce the important notion of ***path-symmetry***.

Definition 1 (Path-Symmetry) *Nodes v_j and v_k are order-L path symmetric with respect to v_i if $P_{i,j}^L = P_{i,k}^L$ and are exact order-l path symmetric w.r.t. v_i if $P_{i,j}^L|_l = P_{i,k}^L|_l$. A set of nodes is (exact) path-symmetric w.r.t v_i if each node in the set is (exact) path-symmetric w.r.t. v_i.*

Within the context of structure learning, path-symmetric sets of nodes correspond to what we denote as ***abstract concepts*** and correspond to collections of entities that have similar neighbourhoods in the hypergraph.

Remark 1 *A necessary condition for nodes v_j and v_k to be order-L path-symmetric w.r.t. v_i is that they are* **order-L distance symmetric** *w.r.t. v_i, i.e. $h_{i,j}^L = h_{i,k}^L$.*

It is computationally infeasible to compute $h_{i,j}^L$ and $P_{i,j}^L$ exactly for large hypergraphs. However, they can both be well-approximated by sampling by running N random walks of length L from node v_i and recording the number of times v_j is hit (Sarkar, Moore, and Prakash 2008). We denote by $\hat{h}_{i,j}^{L,N}$, and by $\hat{P}_{i,j}^{L,N}$, the obtained estimates and refer to them as (L, N) estimates. Finally, we denote by $\hat{C}_{i,j}^{L,N}(\sigma)$ the function from a signature σ in $\mathcal{S}_{i,j}^L$ to the number of occurrences of σ in the paths from v_i to v_j that are encountered while running N random walks of length L. We refer to $\hat{C}_{i,j}^{L,N}(\sigma)$ as the ***L-path signature counts***.

[1] https://github.com/jonathanfeldstein/PRISM

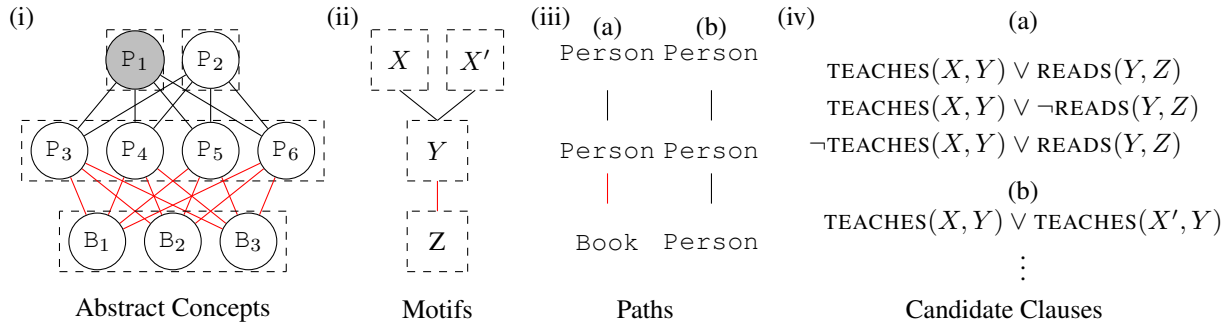

Figure 1: Example Structure-Learning Pipeline: The above shows a dataset about a university class. Nodes $\mathtt{P}_i$ are entities of type person, while $\mathtt{B}_i$ are entities of type book. Black edges represent TEACHES(person, person), and red edges represent READS(person, book): (i) The resulting abstract concepts when random walks are run from source node $\mathtt{P}_1$. Dashed boxes represent concepts, which intuitively are teachers, $\{\mathtt{P}_1\}$, colleagues $\{\mathtt{P}_2\}$, students $\{\mathtt{P}_3, \mathtt{P}_4, \mathtt{P}_5, \mathtt{P}_6\}$ and books $\{\mathtt{B}_1, \mathtt{B}_2, \mathtt{B}_3\}$ (ii) the resulting structural motif, (iii) paths found in the motif (iv) mined candidate clauses.

We denote by $\hat{C}^{L,N}_{i,j}|_l(\sigma)$ the marginal count when only contributions from paths of length exactly $l \in \{1, 2, \ldots, L\}$ are considered. For readability purposes, we will drop the term signature and refer simply to L-***path distributions*** and to L-***path counts***. By path, we will refer to a path signature, unless stated otherwise.

2.1 Example of a Structure Learner: LSM

To illustrate structure learning, we present an overview of the LSM algorithm (Kok and Domingos 2010). The algorithm proceeds in three main steps (denoted S1, S2 and S3 below). The resulting pipeline is summarised in Fig 1.

S1: Finding Structural Motifs Nodes with similar environments in the database hypergraph are first clustered together into *abstract concepts*. Clustering is achieved by running many random walks from each node in the hypergraph. Nodes are then partitioned into sets of path-symmetric nodes based on the similarity of their L-path counts. Each path-symmetric sets then corresponds to an abstract concept.

Example 1 (Abstract Concepts) *In Fig 1, we see that* $\mathtt{P}_1$ *and* $\mathtt{P}_2$ *are both teaching* $\mathtt{P}_3$, $\mathtt{P}_4$, $\mathtt{P}_5$ *and* $\mathtt{P}_6$. *Furthermore,* $\mathtt{P}_3$, $\mathtt{P}_4$, $\mathtt{P}_5$ *and* $\mathtt{P}_6$ *are all reading* $\mathtt{B}_1$, $\mathtt{B}_2$ *and* $\mathtt{B}_3$. *Even though we have not explicitly defined the notion of student, teacher, and book we have that* $\mathtt{P}_3$, $\mathtt{P}_4$, $\mathtt{P}_5$ *and* $\mathtt{P}_6$ *are all path-symmetric w.r.t to* $\mathtt{P}_1$ *and w.r.t* $\mathtt{P}_2$, *as are* $\mathtt{B}_1$, $\mathtt{B}_2$ *and* $\mathtt{B}_3$. *The abstract concepts that we obtain are thus* $\{\mathtt{P}_3, \mathtt{P}_4, \mathtt{P}_5, \mathtt{P}_6\}$, $\{\mathtt{P}_1, \mathtt{P}_2\}$, *and* $\{\mathtt{B}_1, \mathtt{B}_2, \mathtt{B}_3\}$, *which intuitively represent the idea of students, teachers and books, respectively.*

Once abstract concepts are found, they are then joined by the edges that connect them to form *structural motifs*, see Fig 1 (ii). It is the identification of these structural motifs that effectively speeds up the subsequent rule-finding by reducing the search for candidate clauses (c.f. S2). In LSM, computing motifs requires setting six independent hyper-parameters: N the number of random walks ran, L the length of each random walk, θ_{hit} a threshold to select only 'nearby' nodes to the source node of the random walk (those with $\hat{h}^{L,N}_{i,j} \leq \theta_{hit}$), θ_{sym} a threshold for merging nodes based on the similarity of their THTs (all nodes v_j and v_k with $|\hat{h}^{L,N}_{i,j} - \hat{h}^{L,N}_{i,k}| < \theta_{sym}$ are merged), θ_{JS} a threshold for merging nodes by path similarity based on the Jensen-Shannon divergence of their path distributions, and n_{top} the number of paths to consider (in order of descending frequency) when computing the Jensen-Shannon divergence.

S2a: Finding Paths in Motifs Based on the found motifs, sequences (paths) of ground literals that often appear together in the data are generated, see Fig 1 (iii). The fact that the literals appear often together points to the fact that they are likely to be logically dependent on one another.

S2b: Evaluating Candidate Clauses The sequences of ground literals are used to generate candidate clauses. Each clause is evaluated using a likelihood function. The best clauses are then added to the structure-learnt MLN.

S3: Learning the Weights of Candidate Clauses Finally, the algorithm finds the weights of the chosen clauses by maximum-likelihood estimation. This yields a set of formula-weight pairs which define the final MLN.

3 Principled Motif-Finding

Hyperparameter tuning can be one of the most time-costly stages when applying algorithms to real problems. This holds particularly in the case of LSM, where we have six heuristic hyperparameters, as detailed in Section 2.1. In our work, we devise an alternative motif-finding algorithm (PRISM) that depends on only two *intuitive* hyperparameters, thus greatly speeding up the workflow.

3.1 Introducing PRISM

In overview, the steps taken by PRISM are:

For each node v_i in $\mathcal{H}$: (i) run an *optimal* number of random walks originating from v_i and compute, for each $v_j \neq v_i$, the THT estimate $\hat{h}^{L,N}_{i,j}$ and path distribution estimate $\hat{P}^{L,N}_{i,j}$; (ii) partition the nodes $V \in \mathcal{H}$ into sets $A_1, A_2, \ldots, A_M$, that are *statistically significant* order-L distance-symmetric w.r.t. v_i, by merging nodes if the difference in their THTs is below a statistical threshold θ_{sym}.

We describe how to set θ_{sym} in Section 3.3; (iii) further partition the nodes within each A_m into *statistically significant* order-L path-symmetric sets. An algorithm achieving this in $\mathcal{O}(n \ln n)$ (vs $\mathcal{O}(n^3)$ in SOTA) is presented later. Notice that step (ii) serves to reduce the computational cost of step (iii) by applying heuristics that classify the nodes into sets that are most likely to be path-symmetric.

The question remains how to define 'optimal' and 'statistically significant' in the above pipeline. To this end, we introduce two independent parameters, ε to optimise the number of random walks, and α to control the statistical significance threshold of the similarity measure.

3.2 ε-uncertainty: Controlled Path Sampling

Motivation To find good motifs we need to identify abstract concepts. To do this, we compare the path distributions of nodes in the hypergraph representation of the database. However, in real-world applications, computing these distributions exactly is infeasible, so we resort to approximating them through sampling by running random walks. The uncertainty in these approximations will depend on the length L, and number N of random walks. Here we formally define a measure of uncertainty and show how it can be used to set an optimal number of random walks.

Definition 2 (ε-uncertainty) *The uncertainty of the (L, N)-estimate of $h_{i,j}$ is defined by $|h^L_{i,j} - \hat{h}^{L,N}_{i,j}|/h^L_{i,j}$. The uncertainty of the (L, N)-estimate of $P^L_{i,j}$ is defined as the maximum of $|P^L_{i,j}(\sigma) - \hat{P}^{L,N}_{i,j}(\sigma)|/P^L_{i,j}(\sigma)$ among all paths σ in the domain of $P^L_{i,j}$.*

ε-uncertainty is of major importance to the overall theory-induction pipeline as it determines the confidence in the similarity measure between nodes and, ultimately, of the induced theories; the lower the ε, the higher the confidence in the relatedness of nodes. However, naturally, there is a trade-off, as we show below (Thm. 1), as lower uncertainty implies a polynomially higher computational cost. For a given ε we thus seek to find the least number of random walks N that guarantees this uncertainty level. We say that such an N is *ε-optimal*:

Definition 3 (ε-optimality) *N is ε-optimal on $\mathcal{H}$ under L if it is the smallest integer so that for any pair of nodes v_i, v_j in $\mathcal{H}$, the expectation of the uncertainties of (L, N)-estimates of $h_{i,j}$ and $P_{i,j}$ are upper bounded by ε.*

Minimising N is crucial as running random walks is computationally intensive, especially in large hypergraphs.

Usage In Theorem 1 below, we state how to set N to guarantee ε-optimality (for all theorem proofs, see the Appendix on arXiv[2]).

Theorem 1 *An upper bound on the ε-optimal number of random walks N on $\mathcal{H}$ under L is given by*

$$\max\{(L-1)^2/4\varepsilon^2, P^*(\gamma + \ln P^*)/\varepsilon^2\} \qquad (1)$$

where $P^* = 1 + e\left(e^L - 1\right)/(e-1) \gg 1$, e is the number of unique edge labels in $\mathcal{H}$, and $\gamma \approx 0.577$ is the Euler-Mascheroni constant.

[2]https://arxiv.org/pdf/2302.04599

In PRISM, N is automatically computed according to Theorem 1 based on a user-specified ε. In the above, we assumed a fixed L. A good value for L is the diameter of the hypergraph to guarantee that every node can be hit during random walks. In Section 4.2 we revise this assumption and show how L can be reduced based on the properties of $\mathcal{H}$.

3.3 α-significance: Controlled Softness of Formulae

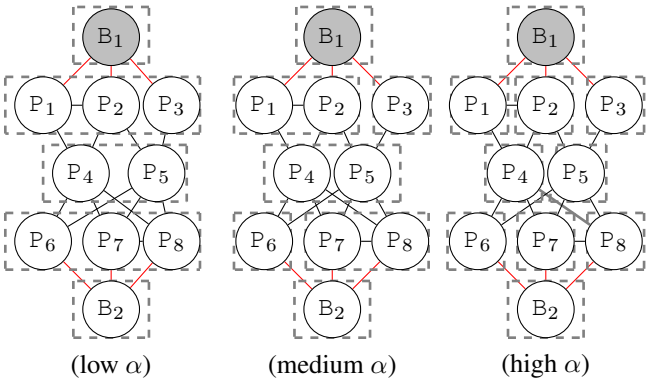

Figure 2: Influence of α: As α increases, the criterion for clustering nodes by path similarity becomes stricter. The source node, $\texttt{B}_1$, is shaded in grey. Nodes that were previously clustered become partitioned. For example: $\{P_1, P_2, P_3\} \rightarrow \{P_1, P_2\}\{P_3\} \rightarrow \{P_1\}\{P_2\}\{P_3\}$.

Motivation Theorem 1 allows us to run the minimum number of walks needed to compute good enough estimates of the truncated hitting times and the path distributions. Based on these estimates, the next step is to partition our data into path-symmetric sets. If we choose very strict standards for clustering nodes together, i.e. only clustering if they have identical truncated hitting times and path distributions, the formulae we find will be very stringent and will not generalize well on unseen data (overfitting). However, if we are too loose on the criteria to merge nodes, the rules we obtain will be too soft and not informative enough (underfitting).

Our approach to controlling rule softness is to introduce statistical tests to decide when two nodes are distance- and path-symmetric and a user-specified parameter $0 < \alpha < 1$. α is the statistical significance level at which two nodes are considered distance- and path-symmetric. α, therefore, measures how lenient we are in merging entities into abstract concepts and by extension an indirect measure of the softness of rules. The effect of changing α for path-symmetry clustering on an example hypergraph is shown in Fig 2.

This approach results in three major benefits:

(i) α is the only clustering parameter compared to the four parameters in SOTA. (ii) α is by construction dataset independent, thus simplifying hyperparameter tuning compared to SOTA. (iii) the α parameter has a direct and intuitive effect on the size of the path-symmetric clusters, with smaller α leading to less-strict statistical tests that ultimately favour finding fewer, but larger path-symmetric sets and thus fewer, more-approximate abstract concepts.

Usage Given truncated hitting times $\hat{h}_{i,j}^{L,N}$, we merge nodes if the difference between their THTs is below a threshold $\theta_{sym}(\alpha)$. Next, given path distributions $\hat{P}_{i,j}^{L,N}$, we propose a hypothesis test to validate whether a set of sampled distributions are statistically similar. We show that both tests can be performed to a specified level of statistical significance given by just one parameter: α.

First, we consider the null hypothesis that nodes v_j and v_k are order-L distance-symmetric w.r.t. v_i, using $|\hat{h}_{i,j}^{L,N} - \hat{h}_{i,k}^{L,N}|$ as a test statistic:

Theorem 2 (Distance-Symmetric Hypothesis Test) *The null hypothesis is rejected at significance level α, i.e. nodes v_j and v_k are not order-L distance-symmetric, if $|\hat{h}_{i,j}^{L,N} - \hat{h}_{i,k}^{L,N}| > ((L-1)/\sqrt{2N})t_{\alpha/2,N-1}$, where $t_{\alpha/2,N-1}$ is the inverse-survival function of an $N-1$ degrees of freedom student-t distribution evaluated at $\alpha/2$.*

Theorem 2 allows us to set parameter θ_{sym} dynamically for each pair of nodes whose hitting times are being compared, such that nodes are merged only if they are distance-symmetric at significance level α:

$$\theta_{sym} = \frac{L-1}{\sqrt{2N}} t_{\alpha/2,N-1}. \quad (2)$$

To measure the degree of *path* symmetry (as opposed to *distance* symmetry), a threshold can be set using a different hypothesis test but based on the same α used to set θ_{sym} above. In the next section, we detail this hypothesis test.

Algorithm 1: PathSymmetricClustering

Input: A, nodes to partition into order-L path-symmetric sets w.r.t. v_i, where $v_i \notin A$
Output: $B_1, \ldots, B_K$, path-symmetric sets
Parameters: α, $\delta = 2$ (parameters N, L are implicit)
if *A is path symmetric at significance level α for each $l \in \{L, L-1, \ldots, 1\}$* **then**
  return $\{A\}$; // Thm. 3
else
  for each $v_{\ell'} \in A$ **compute** & **standardise** $\hat{C}_{i,\ell'}^{L,N}$
  reduce all the $\hat{C}_{i,\ell'}^{l,N}$'s into δ-dimensional feature vectors using PCA
  $Partition \leftarrow \emptyset$
  $RemainingSets \leftarrow \{A\}$
  while $RemainingSets$ *not* ***empty*** **do**
    $S \leftarrow RemainingSets$.pop
    partition S into $\{B_1, B_2\}$ via unsupervised clustering of the $\hat{C}_{i,\ell'}^{l,N}$'s
    for $B_i \in \{B_1, B_2\}$ **do**
      if *B_i is path symmetric at significance level α for each $l \in \{L, L-1, \ldots, 1\}$* **then**
        $Parition$.append(B_i)
      else
        $RemainingSets$.append(B_i)
  return $Partition$

4 Efficient Structural Motif Finding

Above we have discussed how to set our parameters in a principled fashion. In this section, we discuss how to use these parameters in an efficient algorithm (Sec. 4.1), and then further improve speed by reducing the required length (Sec. 4.2) and number of random walks (Sec. 4.3).

4.1 An Improved Path-Symmetry Clustering Algorithm

In this section, we outline an efficient algorithm, which we refer to as PathSymmetricClustering, for partitioning nodes into sets that are path-symmetric at significance level α. This algorithm has $\mathcal{O}(n \ln n)$ complexity in the number of nodes to cluster, which offers a significant improvement over the $\mathcal{O}(n^3)$ complexity of SOTA.

Using the notation introduced in Section 3, we partition each distance-symmetric node set $A_m \in \{A_1, \ldots, A_M\}$ into path-symmetric sets w.r.t. a node v_i. PathSymmetricClustering treats the path counts of the nodes within each A_m as points in a multi-dimensional space of the path signatures. For each A_m, PathSymmetricClustering then clusters nodes into path-symmetric sets as follows: First, we run a hypothesis test (Thm. 3, discussed below) on A_m to check whether the entire set of nodes is path-symmetric at significance level α. If the test passes, all nodes are clustered together. If the test fails, we proceed with recursive clustering (see Alg. 1):

1. Standardize (zero mean, unit variance) the path counts and use PCA to map these standardized counts to a two-dimensional space.
2. Cluster nodes in the reduced space into two sets using unsupervised BIRCH clustering (Zhang, Ramakrishnan, and Livny 1996) .
3. Perform a path-symmetry hypothesis test (Thm. 3) separately on the two identified sets.
4. Clusters failing the test have their nodes repeatedly repartitioned into two new clusters using steps 2 and 3 until all sets of clusters pass the hypothesis test. The output is a set of clusters $\{B_1, B_2, ..., B_k\}$ partitioning the set A_m.

Similar to Section 3.3, the partitioning process in PathSymmetricClustering is driven by a statistical hypothesis test. This time, the desired null hypothesis is that *all the nodes* in a cluster B_k are order-L path-symmetric. Specifically, we test for each cluster the following: that for each $l \in \{L, L-1, \ldots, 1\}$ the cluster B_k is *exact* order-l path-symmetric w.r.t. v_i at significance level α. We denote this null hypothesis H_0.

If H_0 is true, then at significance level α there exists a multinomial distribution, common to all nodes in B_k, from which the empirical exact path counts $\hat{C}_{i,j}^{L,N}|_l$ are drawn. Extending a version of the χ^2 test, we show the following:

Theorem 3 (Path-Symmetric Hypothesis Test) *Let Λ_l be the total number of different paths of length l over all $\mathcal{S}_{i,j}^{L}$'s.*

The null hypothesis that the nodes B_k are order-l exact path symmetric is rejected at significance level α if the statistic

$$Q(B_k) := \sum_{\lambda=0}^{\Lambda_l} \sum_{v_j \in B_k} \left(c_\lambda - c_\lambda^{(j)} \right)^2 \tag{3}$$

exceeds $\chi^2_{\boldsymbol{w},\boldsymbol{\nu}}(\alpha)$, where

$$c_\lambda^{(j)} := \hat{C}_{i,j}^{L,N}(\lambda), \quad c_0^{(j)} := N - \sum_{\lambda=1}^{\Lambda_l} c_\lambda^{(j)}, \tag{4}$$

$$c_\lambda := \frac{1}{|B_k|} \sum_{v_j \in B_k} c_\lambda^{(j)},$$

and $\chi^2_{\boldsymbol{w},\boldsymbol{\nu}}(\alpha)$ is a generalised chi-squared distribution, weight parameters $\boldsymbol{w}$ and degree of freedom parameters $\boldsymbol{\nu} = (1, 1, ..., 1)$, evaluated at significance level α. Above $\boldsymbol{w}$ are the eigenvalues of the block matrix $\tilde{\boldsymbol{\Sigma}}$ with components

$$\tilde{\Sigma}_{\lambda,\lambda'}^{(b,b')} = N \left(\delta_{b,b'} - \frac{1}{|B_k|} \right) \cdot$$
$$\left(\delta_{\lambda,\lambda'} \frac{c_\lambda}{N} \left(1 - \frac{c_\lambda}{N} \right) - \left(1 - \delta_{\lambda,\lambda'} \right) \frac{c_\lambda}{N} \frac{c'_\lambda}{N} \right),$$

where $b, b' \in (1, \ldots, |B_k|)$ index blocks, $\lambda, \lambda' \in (0, \ldots, \Lambda_l)$ index within blocks and $\delta_{\lambda,\lambda'}$ is the Kronecker delta.

Remark 2 *By requiring knowledge of the eigenvalues $\boldsymbol{w}$, Theorem 3 suggests that an eigendecomposition of $\tilde{\boldsymbol{\Sigma}}$ is necessary. However, we show in the Appendix how this calculation can be avoided by approximating the generalised chi-squared distribution by a gamma distribution.*

4.2 Running Shorter Random Walks

We now show how to set a minimal L needed for the random walks to span areas of interest in the hypergraph. We do this by hierarchical clustering.

Motivation We implement hierarchical clustering by iteratively cutting the hypergraph along sparse cuts until no sparse cuts remain. This algorithm results in three benefits: (i) Splitting the hypergraph into smaller sub-graphs leads to smaller diameters and therefore smaller L. This, by extension, also reduces N (Thm. 1). (ii) Having fewer nearby nodes means that the subsequent partitioning in PathSymmetricClustering is faster. (iii) Hierarchical clustering identifies groups of densely-connected nodes, which helps us to ignore spurious links. Spuriously-connected nodes appear rarely in the path signatures and therefore only add noise to the path signature counts. By focusing random walks on areas of interest, we are hitting nodes that are densely connected more often and gaining more accurate statistics of truncated hitting times and empirical path distributions.

Hierarchical Clustering Algorithm The algorithm HClustering is based on spectral clustering, a standard approach for cutting graphs along sparse cuts. A discussion of spectral clustering is beyond the scope of this paper. Note that the is no equivalent approach for hypergraphs, so we propose to translate a hypergraph into a graph and then perform spectral clustering as follows:

In overview, HClustering begins by converting a hypergraph $\mathcal{H} = (V, E)$ into a weighted graph $\mathcal{G}$ by expanding cliques over each hyperedge. Next, $\mathcal{G}$ is recursively bipartitioned using the sweep set approximation algorithm for the Cheeger-cut (Chang, Shao, and Zhang 2017). The result of the partitioning is a set of subgraphs $G := \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_k\}$. The partitioning terminates whenever the second-smallest eigenvalue of the symmetric Laplacian matrix λ_2 exceeds a threshold value λ_2^{max}. λ_2^{max} is dataset independent and thus fixed in our implementation. Finally, each subgraph $\mathcal{G}_i$ is then converted into a hypergraph $\mathcal{H}_i = (V_i, E_i)$ such that the vertex set V_i of the hypergraph is initialised to be the vertex set of $\mathcal{G}_i$. The edge set E_i is then constructed by adding all hyperedges $e \in E$ whose strict majority of element vertices appear in V_i, i.e. $E_i := \{e \in E \mid |e \cap V_i| > |e|/2\}$. As a consequence, no nodes nor edges are lost during clustering. HClustering returns the set of sub-hypergraphs $\{\mathcal{H}_1, \mathcal{H}_2, ..., \mathcal{H}_k\}$. After partitioning, we run the rest of the pipeline with L set to the diameter of each $\mathcal{H}_i$.

Our entire pipeline for learning abstract concepts from a relational database is summarised in Algorithm 2.

Algorithm 2: PRISM

Input: $\mathcal{H}$, the hypergraph representation of the input relational database
Output: Path-symmetric sets (abstract concepts $\mathcal{C}$) of nodes w.r.t. each v_i in $\mathcal{H}$
Parameters: ε, α
$\mathcal{H}_1, \ldots, \mathcal{H}_K :=$ HClustering($\mathcal{H}$) ; `// Sec. 4.2`
Let V_k denote the set of nodes in $\mathcal{H}_k$
for $1 \leq k \leq K$ **do**
  set L to the diameter of $\mathcal{H}_k$
  compute ε-optimal N on $\mathcal{H}_k$ under L ; `// Th.1`
  for ***each*** *node v_i in $\mathcal{H}_k$* **do**
    for each $v_j \neq v_i$ in $\mathcal{H}_k$ compute $\hat{P}_{i,j}^{L,N}$ and $\hat{h}_{i,j}^{L,N}$; `// Sec. 2`
    partition V_k into distance-symmetric sets $\{A_1, A_2, ..., A_M\}$ using α-significance ; `// Th. 2, Sec. 2`
    for $1 \leq m \leq M$ **do**
      $\mathcal{C}_m :=$ PathSymmetricClustering(A_m, α) ; `// Th. 3, Sec. 4`
return *all $\mathcal{C}_m$'s*

4.3 Running Fewer Random Walks

As a final optimization step, we comment on how the number of random walks can be further reduced. The number of walks as implied by Theorem 1 can be very large since P^* grows exponentially with L. Therefore in practice, rather than running enough walks to guarantee ε-boundedness for all path signatures, we only run enough walks to guarantee ε-boundedness for the top k most common path signatures.

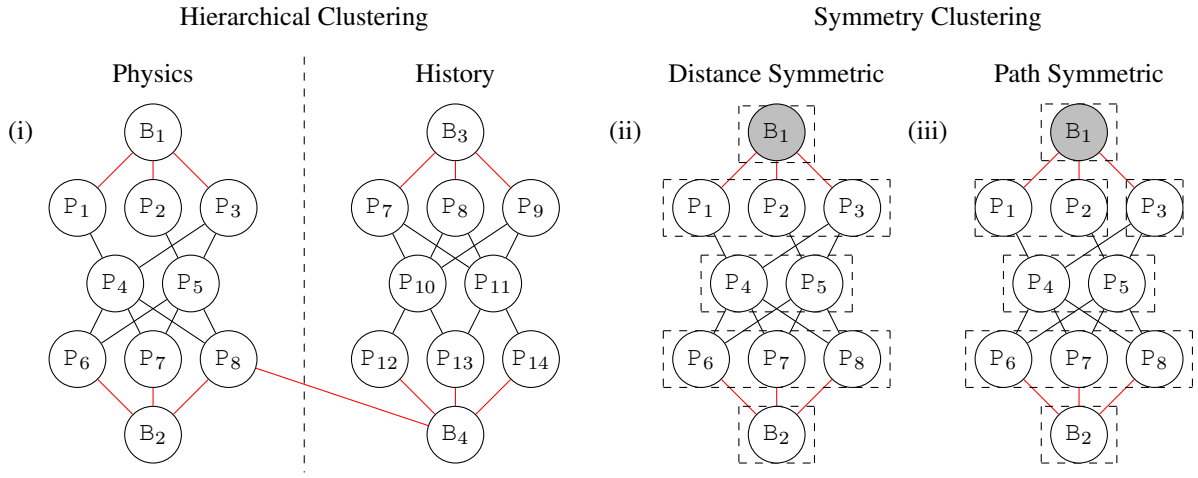

Figure 3: PRISM Pipeline: A visual example of the PRISM algorithm applied to an academic departments toy dataset. Nodes $\texttt{P}_i$ are entities of type `person`, while $\texttt{B}_i$ are entities of type `book`. Black edges represent TEACHES(`person`, `person`), and red edges represent READS(`person`, `book`) predicates. Although not explicitly annotated in the data, entities $\{\texttt{P}_4, \texttt{P}_5, \texttt{P}_{10}, \texttt{P}_{11}\}$ are in fact professors and the remaining `person` entities are students. *Hierarchical Clustering Preprocessing*: The physics and history departments are connected by a single spurious link. In this example, hierarchical clustering therefore stops after one iteration, cutting along the departments (dotted line). To avoid information loss, the spurious link between $\texttt{P}_8$ and $\texttt{B}_4$ will be preserved in one of the clusters (Sec. 4.2). *Symmetry Clustering*: Here we focus on the left sub-graph obtained from hierarchical clustering in (i). Running random walks from $\texttt{B}_1$, we show examples of the distance-symmetric and path-symmetric clusters that we obtain in (ii) and (iii), respectively. Note how $\{\texttt{P}_1, \texttt{P}_2, \texttt{P}_3\}$ in (ii) is partitioned into $\{\texttt{P}_1, \texttt{P}_2\}$ and $\{\texttt{P}_3\}$ in (iii) since path-symmetry is more stringent than distance symmetry.

Theorem 4 (Fewer Random Walks) *An upper bound on N sufficient for the k^{th} most probable path to have uncertainty less than or equal to ε is*

$$N = \frac{(k+1)\left(\gamma + \ln P^*\right) - 1}{\varepsilon^2}.$$

In our implementation, we use $k = 3$ since we deem it to be the smallest value (and therefore requiring the fewest random walks) that still allows for meaningful comparison between path distributions.

5 Extended Example

We now illustrate the entire PRISM pipeline through an extended example, shown in Fig 3. In the figure, we consider the hypergraph representation of a dataset describing a physics and history department in a university, containing two types of entities (`person` and `book`) and two relations (TEACHES(`person`, `person`) and READS(`person`, `book`)).

The first stage of PRISM applies hierarchical clustering to the dataset to identify densely-connected nodes. In this example, the physics and history departments are only connected by a single, spurious link, and the hierarchical clustering stops after one iteration, cutting along the departments. In addition, we can verify here that the hierarchical clustering results in an almost two-fold computational speed-up: The original hypergraph in Fig 3(i) has diameter 9. If we set $\varepsilon = 0.1$, then Theorem 4 gives an upper bound on the ε-optimal number of random walks for accurate path distributions of $N = 3.0 \times 10^3$. After hierarchical clustering, the hypergraphs have diameter 4, and Theorem 4 for the same ε gives an upper bound of $N = 1.6 \times 10^3$.

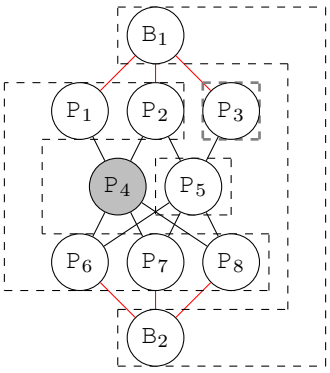

Figure 4: Changing the Source Node: In this case, the source node is $\texttt{P}_4$ (a professor) and we obtain a different, although intuitive partitioning: $\texttt{P}_5$ is a colleague of $\texttt{P}_4$, $\{\texttt{P}_1, \texttt{P}_2, \texttt{P}_6, \texttt{P}_7, \texttt{P}_8\}$ are $\texttt{P}_4$'s students, $\texttt{P}_3$ is a student that $\texttt{P}_4$ is not teaching and $\{\texttt{B}_1, \texttt{B}_2\}$ are the academic books of the department. Note how it is possible to extract these abstract concepts, even though the only explicit information we initially provided was that entities are either books or people.

After hierarchical clustering, PRISM applies symmetry clustering in two stages: first by identifying distance-symmetric sets based on their truncated hitting times, then by identifying path-symmetric nodes within these sets based

	Algorithm	AUC	CLL	ACC	MF TIME (s)	SL TIME (s)
IMDB	PRISM	**0.141 ± 0.027**	**-0.18 ± 0.03**	**0.84 ± 0.02**	**0.086 ± 0.018**	320 ± 40
	LSM	0.12 ± 0.03	-0.25 ± 0.06	0.78 ± 0.04	1.25 ± 0.10	430 ± 20
	BOOSTR	0.062 ± 0.013	-0.69 ± 0.006	0.504 ± 0.004	N/A	**165.7 ± 129**
UWCSE	PRISM	**0.402 ± 0.028**	**-0.0098 ± 0.0009**	**0.993 ± 0.002**	**0.40 ± 0.06**	640 ± 350
	LSM	0.392 ± 0.023	**-0.0098 ± 0.0009**	0.992 ± 0.002	4.23 ± 0.80	3140 ± 270
	BOOSTR	0.0098 ± 0.003	-2.114 ± 0.004	0.121 ± 0.001	N/A	**30.5 ± 3.9**
WEBKB	PRISM	**0.57 ± 0.04**	**-0.0092 ± 0.0011**	**0.991 ± 0.002**	**0.118 ± 0.038**	102 ± 5
	LSM	**0.57 ± 0.04**	**-0.0092 ± 0.0011**	**0.991 ± 0.002**	2.5 ± 0.4	220 ± 10
	BOOSTR	0.0335 ± 0.0021	-2.14 ± 0.09	0.118 ± 0.010	N/A	**9.3 ± 0.4**

Table 1: Area Under the Precision Recall Curve (AUC), Conditional Log Likelihood (CLL), Accuracy (ACC), Motif Finding (MF) time, and Structure Learning (SL) time comparisons of PRISM, LSM and BOOSTR on three datasets.

on path distributions. The first stage only serves to speed up the subsequent path-symmetric clustering since path-symmetry implies distance symmetry (Rmk. 1), but checking distance symmetry is quicker ($\mathcal{O}(n)$ vs $\mathcal{O}(n \ln n)$ for PathSymmetricClustering). Note that, in this example, the source node was chosen as B_1 and the hypergraph has a high degree of symmetry relative to the source node, which explains why the distance-symmetric and path-symmetric sets are almost identical (Fig. 3 (ii) and (iii)). For more realistic datasets, where global symmetries in a hypergraph are rare, the differences between distance-symmetric and path-symmetric clustering will be more pronounced.

We finish this section by illustrating the effect of changing the source node of the random walks. Recall that sets of symmetric nodes, i.e., the abstract concepts, are always found with respect to a specific source node. Changing the source node, therefore, changes the learnt concepts. This idea is illustrated in Fig 4, where the source node is changed from B_1 to P_4, resulting in different clusterings. When random walks were run from B_1 we obtained the familiar concepts of teachers, colleagues, students and books. However, Fig 4 illustrates how abstract concepts can often be less intuitive, but still illustrate subtle relationships in the data. In PRISM we run random walks from each node in the hypergraph in turn. This helps to identify a wide range of abstract concepts.

6 Experiments

We compare our motif-finding algorithm, PRISM, against the current state-of-the-art, LSM (Kok and Domingos 2010) and BOOSTR (Khot et al. 2015), in terms of speed and accuracy of the mined MLNs.

Datasets We used benchmark datasets adopted by the structure learning literature: UW-CSE (Richardson and Domingos 2006), IMDB, and WEBKB. The IMDB dataset is subsampled from the IMDB.com database and describes relationships among movies, actors and directors. The UW-CSE dataset describes an academic department and the relationships between professors, students and courses. The WEBKB consists of Web pages and hyperlinks collected from four computer science departments. Each dataset has five splits. Each time we used one split to test accuracy and the remaining splits for training. The reported results are the average over all five permutations.

Problem Given a dataset with partial observations, we want to predict the truth values for unobserved data. For example, for the IMDB dataset we might not know every actor who starred in a movie. We then predict, for each actor in the database, the likelihood of an actor to have starred in a given movie. We remark that our unobserved data spans across every possible predicate in the database, e.g. for IMDB this would include STARRINGIN(movie,person), ACTOR(person)... This problem thus reduces to predicting missing edges in the hypergraph.

Baseline and Evaluation We used the entire LSM pipeline (Kok and Domingos 2010) as a baseline. We used the lifted belief propagation inference tool of Alchemy (Kok et al. 2007) to calculate the averaged conditional log-likelihood on each entity (ground atom) in the test split. For LSM, we used the same hyperparameters as originally adopted by the authors (Kok and Domingos 2010). In addition, we compared our work to the authors' publically available implementation of BOOSTR (Khot et al. 2015).

Experiment Setup Firstly, we ran PRISM and then, the remainder of the unmodified LSM pipeline. We used $\varepsilon = 0.1$ and $\alpha = 0.01$ throughout, as both are dataset-independent. We ran all experiments on a desktop with 32Gb of RAM and a 12-core 2.60GHz i7-10750H CPU.

Metrics We used standard measures from the structure learning literature. In particular, we measured accuracy and conditional log-likelihood for all datasets, as well as the area under the precision-recall curve (AUC) as it provides a more robust measure. The runtimes of the motif-finding step and the overall structure learning time are also reported.

Results In Table 1, we see that compared to LSM, we improve in accuracy on the IMDB dataset by 6%, while on UW-CSE and WEBKB the improvement is negligible. This is because LSM already found rules that generalized the data extremely well. However, as expected, the runtime of our algorithm is significantly reduced for all datasets. For motif finding, we see that our optimised algorithm is 10x-20x faster than LSM's motif-finding time. The overall structure learning computation is up to 5x faster than LSM. This is despite the main computational bottleneck for structure learning occurring during rule induction and evaluation - parts of

the pipeline that were left unmodified. This suggests that our algorithm more tightly constrains the subsequent rule induction by finding more accurate motifs, thereby giving a speed improvement in these areas of the pipeline too.

We are slower compared to BOOSTR. However, PRISM's accuracy drastically improves over BOOSTR. We believe the differences in time and accuracy between datasets for BOOSTR stem from the quality of the background knowledge: while on IMDB and UW-CSE background knowledge was given, on WEBKB it was not. No background knowledge was provided to LSM or PRISM.

7 Related Work

In this section, we will review prior art in structure learning across a variety of logical languages. As we show below, every one of these approaches is based on learnt or user-defined templates to restrict the search space of candidate formulae. These templates are exactly the motifs that we are finding automatically and efficiently with the proposed framework.

ILP To alleviate the need to manually provide logical theories, several communities have developed techniques for inducing logical theories. One of the most influential family techniques for mining Horn clauses is that of *Inductive Logic Programming* (ILP), e.g., FOIL (Quinlan 1990), MDIE (Muggleton 1995) and Inspire (Schüller and Benz 2018). Recently, Evans and Grefenstette proposed a differentiable variant of ILP (Evans and Grefenstette 2018) to support the mining of theories in noisy settings. ILP techniques require users to provide in advance the patterns of the formulas to mine, as well as to provide both positive and negative examples. The above requirements, along with issues regarding scalability (Evans and Grefenstette 2018), restrict the application of ILP techniques in large and complex scenarios. Our work specifically focuses on finding these patterns automatically and are not restricted to Horn clauses.

Recently, several techniques aim to mine rules in a differentiable fashion. One of them is Neural LP (Yang, Yang, and Cohen 2017), a differentiable rule mining technique based on TensorLog (Cohen, Yang, and Mazaitis 2020). The authors in (Guu, Miller, and Liang 2015) presented a RESCAL-based model to learn from paths in knowledge graphs, while Sadeghian et al. proposed DRUM, a differentiable technique for learning uncertain rules in first-order logic (Sadeghian et al. 2019). A limitation of the above line of research is that they mainly focus on rules of a specific transitive form only. Other techniques for differentiable rule mining have been proposed in (Das et al. 2017; Minervini et al. 2018; Rocktäschel and Riedel 2017). In contrast to this line of work, our motif-finding algorithm helps in pipelines that can find more general first-order logic rules.

MLN The first and somewhat naive structure learning algorithm proposed for MLNs is called *top-down structure learning* (TDSL) (Kok and Domingos 2005). The idea is to perform a near-exhaustive search for candidate logical rules and then construct an MLN by recursively retaining rules that lead to the best improvement in the pseudo-likelihood approximation. That means that the algorithm starts with S2 directly. Due to the exponential search space, the algorithm is not able to find long rules in large datasets and fails to find rules that truly generalize and capture the underlying data. The following approaches in this line of research all prepended the rule generation with a pattern-finding step (S1) - the core of our research. The first paper that proposed such an approach was *bottom-up structure learning* (BUSL) (Mihalkova and Mooney 2007), where the idea was to prespecify template networks akin to our motifs, that would be good candidates for potential rules and iteratively build on these templates to find more complex rules.

To tackle the high computational overhead of structure learning, (Khot et al. 2015) introduce BOOSTR, a technique that simultaneously learns the weights and the clauses of an MLN. The key idea is to transform the problem of learning MLNs by translating MLNs into regression trees and then uses functional gradient boosting (Friedman 2000) along those trees to find clauses. Further, it tries to learn under unobserved data. To this end, they introduced an EM-based boosting algorithm for MLNs. This approach also requires templates, however, they must be user-defined, which requires additional effort and can restrict applications. While showing promising results in terms of runtime, the technique supports only Horn clauses, and its performance drastically decreases in the absence of background knowledge, as we later show in our empirical results (Sec. 6).

The current SOTA in this line of research is *learning through structural motifs* (LSM) (Kok and Domingos 2010), where similar to the template-networks, motifs are identified in the hypergraph representation of the data, by running random walks on the graph and identify symmetric patterns through path signature symmetry of the random walks. The finding of good motifs or templates is the differentiating point between the different algorithms and has been shown to have the most significant impact on the quality of the ultimate rules. A principled, robust and efficient algorithm for finding such motifs could therefore improve these and future algorithms. We believe that we are the first to propose such a principled and efficient algorithm for finding motifs.

8 Conclusion

We made a key step toward learning the structure of logical theories - mining structural motifs. We presented the first principled mining motif technique in which users can control the uncertainty of mined motifs and the softness of the resulting rules. Furthermore, we reduced the overall complexity of motif mining through a novel $\mathcal{O}(n \ln n)$ clustering algorithm. Our empirical results against the state-of-the-art show improvements in runtime and accuracy by up to 80% and 6%, respectively, on standard benchmarks. While we focused on lifted graphical models, our work can be used to learn the formulas of other types of logical theories as well. One interesting direction of future work is to integrate our motif-mining technique with differential rule mining approaches as our empirical analysis shows that purely-symbolic based approaches for that task can sometimes be the bottleneck. A second direction is to integrate our motif-mining approach with Graph Neural Networks and provide a similar formal analysis.

References

Bach, S. H.; Broecheler, M.; Huang, B.; and Getoor, L. 2017. Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research*.

Chang, K.; Shao, S.; and Zhang, D. 2017. Cheeger's cut, maxcut and the spectral theory of 1-Laplacian on graphs. *Science China Mathematics*, 60(11): 1963–1980.

Chen, Y.; and Wang, D. Z. 2014. Knowledge expansion over probabilistic knowledge bases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 649–660. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-4503-2376-5.

Cohen, W. W.; Yang, F.; and Mazaitis, K. 2020. TensorLog: A Probabilistic Database Implemented Using Deep-Learning Infrastructure. *J. Artif. Intell. Res.*, 67: 285–325.

Crane, R.; and Mcdowell, L. K. 2012. Investigating markov logic networks for collective classification. *Proceedings of the 4th International Conference on Agents and Artificial Intelligence*, 1: 5–15.

Das, R.; Dhuliawala, S.; Zaheer, M.; Vilnis, L.; Durugkar, I.; Krishnamurthy, A.; Smola, A. J.; and McCallum, A. 2017. Go for a Walk and Arrive at the Answer: Reasoning Over Paths in Knowledge Bases using Reinforcement Learning. *CoRR*, abs/1711.05851.

Evans, R.; and Grefenstette, E. 2018. Learning Explanatory Rules from Noisy Data. *J. Artif. Intell. Res.*, 61: 1–64.

Friedman, J. H. 2000. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29: 1189–1232.

Getoor, L.; and Taskar, B. 2007. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Guu, K.; Miller, J.; and Liang, P. 2015. Traversing Knowledge Graphs in Vector Space. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 318–327. Lisbon, Portugal: Association for Computational Linguistics.

Ha; Rowe; Mott; and Lester. 2011. Goal Recognition with Markov Logic Networks for Player-Adaptive Games. *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-11)*.

Hu, Z.; Ma, X.; Liu, Z.; Hovy, E.; and Xing, E. 2016. Harnessing Deep Neural Networks with Logic Rules. In *ACL*, 2410–2420.

Khot, T.; Natarajan, S.; Kersting, K.; and Shavlik, J. 2015. Gradient-based boosting for statistical relational learning: the Markov logic network and missing data cases. *Machine Learning*, 100(1): 75–100.

Kok, S.; and Domingos, P. 2005. Learning the structure of Markov logic networks. In *Proceedings of the 22nd international conference on Machine learning - ICML '05*, 441–448. Bonn, Germany: ACM Press. ISBN 978-1-59593-180-1.

Kok, S.; and Domingos, P. 2010. Learning Markov Logic Networks Using Structural Motifs. *AAAIWS: Proceedings of the 6th AAAI Conference on Statistical Relational Artificial Intelligence*.

Kok, S.; Singla, P.; Richardson, M.; Domingos, P.; Sumner, M.; and Poon, H. 2007. The alchemy system for statistical relational AI: User Manual. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA.

Lippi, M.; and Frasconi, P. 2009. Prediction of protein β-residue contacts by Markov logic networks with grounding-specific weights. *Bioinformatics*, 25(18): 2326–2333.

Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; and De Raedt, L. 2018. DeepProbLog: Neural Probabilistic Logic Programming. In *NeurIPS*, 3749–3759.

Mihalkova, L.; and Mooney, R. J. 2007. Bottom-up learning of Markov logic network structure. In *Proceedings of the 24th international conference on Machine learning - ICML '07*, 625–632. Corvalis, Oregon: ACM Press. ISBN 978-1-59593-793-3.

Minervini, P.; Bosnjak, M.; Rocktäschel, T.; and Riedel, S. 2018. Towards Neural Theorem Proving at Scale. *CoRR*, abs/1807.08204.

Muggleton, S. H. 1995. Inverse Entailment and Progol. *New Gener. Comput.*, 13(3&4): 245–286.

Quinlan, J. R. 1990. Learning Logical Definitions from Relations. *Mach. Learn.*, 5(3): 239–266.

Richardson, M.; and Domingos, P. 2006. Markov logic networks. *Machine Learning*, 62(1-2): 107–136.

Riedel, S.; and Meza-Ruiz, I. 2008. Collective semantic role labelling with Markov logic. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, CoNLL '08, 193–197. USA: Association for Computational Linguistics. ISBN 978-1-905593-48-4.

Rocktäschel, T.; and Riedel, S. 2017. End-to-end Differentiable Proving. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Russell, S. 2015. Unifying logic and probability. *Communications of the ACM*, 58(7): 88–97.

Sadeghian, A.; Armandpour, M.; Ding, P.; and Wang, D. Z. 2019. DRUM: End-To-End Differentiable Rule Mining On Knowledge Graphs. In Wallach, H. M.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E. B.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 15321–15331.

Sarkar, P.; Moore, A. W.; and Prakash, A. 2008. Fast Incremental Proximity Search in Large Graphs. *Proceedings of the 25th International Conference on Machine Learning*, 896–903.

Schüller, P.; and Benz, M. 2018. Best-effort inductive logic programming via fine-grained cost-based hypothesis generation - The inspire system at the inductive logic programming competition. *Mach. Learn.*, 107(7): 1141–1169.

Singla, P.; and Domingos, P. 2006. Entity Resolution with Markov Logic. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, 572–582. USA: IEEE Computer Society. ISBN 978-0-7695-2701-7.

Wang, H.; and Poon, H. 2018. Deep Probabilistic Logic: A Unifying Framework for Indirect Supervision. In *EMNLP*, 1891–1902.

Wu, F.; and Weld, D. S. 2008. Automatically refining the wikipedia infobox ontology. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, 635–644. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-60558-085-2.

Yang, F.; Yang, Z.; and Cohen, W. W. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 2316–2325. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781510860964.

Yang, Z.; Ishay, A.; and Lee, J. 2020. NeurASP: Embracing Neural Networks into Answer Set Programming. In *IJCAI*, 1755–1762.

Zhang, T.; Ramakrishnan, R.; and Livny, M. 1996. BIRCH: an efficient data clustering method for very large databases. *ACM SIGMOD Record*, 25(2): 103–114.