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

Jonathan Feldstein*1,2, Dominic Phillips*1, Efthymia Tsamoura3
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, eft.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 \( O(n \ln n) \) (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
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 $O(n^3)$ 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 $O(n \ln n)$ 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. 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 $\text{label}(e_i)$ to denote the label of the hyperedge $e_i$. A path $\pi$ 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 $(v_0, e_0, v_1, \ldots, e_{L-1}, v_L)$ 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 $\pi$ is the sequence of the labels of the edges occurring in $\pi$, i.e., $(\text{label}(e_0), \ldots, \text{label}(e_{L-1}))$.

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 $h_{i,j}^L$ (THT) is the hitting time where the length of the random walk is limited to at most $L$ steps. It is defined recursively as $h_{i,j}^L = 1 + \sum_{k \neq j} p_{i,k} h_{k,j}^{L-1}$, where $p_{i,j}$ is the transition matrix of the random walk, with $h_{i,i}^L = 0$ if $i = j$, and $h_{i,j}^L = L$ 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 $S_{i,j}^L$ 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 $S_{i,j}^L$ under a given random walk process. The marginal $L$-path signature distribution $P_{i,j}^L|\sigma$ is the marginal probability distribution when only paths of length exactly $\ell \in \{1, 2, \ldots, L\}$ are considered. The quantities $P_{i,j}^L(\sigma)$ and $P_{i,j}^L|\sigma$ respectively denote the probability and marginal probability of path signature $\sigma$. With this, we now introduce the important notion of path-symmetry.

Definition 1 (Path-Symmetry) Nodes $v_i$ and $v_j$ are order-$L$ path symmetric with respect to $v_j$ if $P_{i,j}^L = P_{j,i}^L$ and are exact order-$L$ path symmetric w.r.t. $v_i$ if $P_{i,j}^L = P_{i,j}^L|\ell_i k_i l_i k_i$. 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 $v_i$ and $v_k$ to be order-$L$ path-symmetric w.r.t. $v_i$ 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 $h_{i,j}^L$ and $P_{i,j}^L$ the observed estimates and refer to them as $(L, N)$ estimates. Finally, we denote by $C_{i,j}^L(N)$ the function from a signature $\sigma$ in $S_{i,j}^L$ to the number of occurrences of $\sigma$ in the paths from $v_i$ to $v_j$ that are encountered while running $N$ random walks of length $L$. We refer to $C_{i,j}^L(N)$ as the $L$-path signature counts.
We denote by $\hat{C}_{i,j}^{L,N}(\sigma)$ 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 set then corresponds to an abstract concept.

Example 1 (Abstract Concepts) In Fig 1, we see that $P_1$ and $P_2$ are both teaching $P_3$, $P_4$, $P_5$ and $P_6$. Furthermore, $P_3$, $P_4$, $P_5$ and $P_6$ are all reading $B_1$, $B_2$ and $B_3$. Even though we have not explicitly defined the notion of student, teacher, and book we have that $P_3$, $P_4$, $P_5$ and $P_6$ are all path-symmetric w.r.t to $P_1$ and w.r.t $P_2$, as are $B_1$, $B_2$ and $B_3$. The abstract concepts that we obtain are thus $\{P_3, P_4, P_5, P_6\}$, $\{P_1, P_2\}$, and $\{B_1, B_2, B_3\}$, 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, $\theta_{hit}$ a threshold to select only ‘nearby’ nodes to the source node of the random walk (those with $\hat{h}_{i,j}^{L,N} \leq \theta_{hit}$), $\theta_{sym}$ a threshold for merging nodes based on the similarity of their THTs (all nodes $v_j$ and $v_k$ with $|\hat{h}_{i,j}^{L,N} - \hat{h}_{i,k}^{L,N}| < \theta_{sym}$ are merged), $\theta_{JS}$ 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 $\hat{h}_{i,j}^{L,N}$ and path distribution estimate $\tilde{p}_{i,j}^{L,N}$; (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 $\theta_{sym}$. 

![Diagram](image-url)
We describe how to set $\theta_{\text{sym}}$ 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 $O(n \ln n)$ (vs $O(n^3)$ 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, $\varepsilon$ to optimise the number of random walks, and $\alpha$ to control the statistical significance threshold of the similarity measure.

### 3.2 $\varepsilon$-uncertainty: Controlled Path Sampling

**Motivation** To find good motifs we need to identify abstract concepts. To do this, we compare the path distributions by applying heuristics that classify the nodes into sets that are most likely to be path-symmetric.

**Definition 2 ($\varepsilon$-uncertainty)** The uncertainty of the $(L,N)$-estimate of $h_{i,j}$ is defined by $|h^L_{i,j} - h^{L,N}_{i,j}|/h^L_{i,j}$.

The uncertainty of the $(L,N)$-estimate of $P_{i,j}$ is defined as the maximum of $|P^L_{i,j}(\sigma) - P^{L,N}_{i,j}(\sigma)|/P^L_{i,j}(\sigma)$ among all paths $\sigma$ in the domain of $P^L_{i,j}$.

$\varepsilon$-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 $\varepsilon$, 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 $\varepsilon$ we thus seek to find the least number of random walks $N$ that guarantees this uncertainty level. We say that such an $N$ is $\varepsilon$-optimal:

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

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 $\varepsilon$-optimality (for all theorem proofs, see the Appendix on arXiv).  

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

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

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

In PRISM, $N$ is automatically computed according to Theorem 1 based on a user-specified $\varepsilon$. 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 $H$.

#### 3.3 $\alpha$-significance: Controlled Softness of Formulae

**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 $0 < \alpha < 1$. $\alpha$ is the statistical significance level at which two nodes are considered distance- and path-symmetric. $\alpha$, 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 $\alpha$ for path-symmetry clustering on an example hypergraph is shown in Fig 2.

This approach results in three major benefits:

(i) $\alpha$ is the only clustering parameter compared to the four parameters in SOTA. (ii) $\alpha$ is by construction dataset independent, thus simplifying hyperparameter tuning compared to SOTA. (iii) The $\alpha$ parameter has a direct and intuitive effect on the size of the path-symmetric clusters, with smaller $\alpha$ 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 $P_{i,j}^{L,N}$, 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: $\alpha$.

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 $\alpha$, 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 $\theta_{sym}$ 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 $\alpha$:

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

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 $\alpha$ used to set $\theta_{sym}$ above. In the next section, we detail this hypothesis test.

### 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-Symmetric 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 $\alpha$. This algorithm has $O(n \ln n)$ complexity in the number of nodes to cluster, which offers a significant improvement over the $O(n^3)$ 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 $\alpha$. If the test passes, all nodes are clustered together. If the test fails, we proceed with recursive clustering (see Alg. 1):

#### Algorithm 1: PathSymmetricClustering

1. **Input:** $A$, nodes to partition into order-$L$ path-symmetric sets w.r.t. $v_i$, where $v_i \notin A$
2. **Output:** $B_1, \ldots, B_K$, path-symmetric sets
3. **Parameters:** $\alpha$, $\delta = 2$ (parameters $N, L$ are implicit)
4. **if** $A$ is path symmetric at significance level $\alpha$ for each $l \in \{L, L - 1, \ldots, 1\}$ **then**
5. return $\{A\}$; // Thm. 3
6. **else**
7. for each $v_i \in A$ compute & standardise $\hat{C}_{i,i'}^{L,N}$
8. reduce all the $\hat{C}_{i,i'}^{L,N}$’s into $\delta$-dimensional feature vectors using PCA
9. $\text{Partition} \leftarrow \emptyset$
10. $\text{RemainingSets} \leftarrow \{A\}$
11. while $\text{RemainingSets}$ not empty do
12. $S \leftarrow \text{RemainingSets}.\text{pop}$
13. $\text{partition} S$ into $\{B_1, B_2\}$ via unsupervised clustering of the $\hat{C}_{i,i'}^{L,N}$’s
14. for $B_l \in \{B_1, B_2\}$ do
15. if $B_l$ is path symmetric at significance level $\alpha$ for each $l \in \{L, L - 1, \ldots, 1\}$ **then**
16. $\text{Partition}.\text{append}(B_l)$
17. **else**
18. $\text{RemainingSets}.\text{append}(B_l)$
19. return $\text{Partition}$

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 $\{B_1, B_2, \ldots, B_k\}$ 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 $\alpha$. We denote this null hypothesis $H_0$.

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

**Theorem 3 (Path-Symmetric Hypothesis Test)** Let $S_{i,j}$ be the total number of different paths of length $l$ over all $\hat{C}_{i,i'}^{L,N}$’s.
The null hypothesis that the nodes $B_k$ are order-$l$ exact path symmetric is rejected at significance level $\alpha$ if the statistic

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

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

$$c_{\lambda}^{(j)} := \hat{C}_{i,j} L,N(\lambda), \quad c_{\lambda} := N - \sum_{\lambda=1}^{\Lambda_l} c_{\lambda}^{(j)}, \quad c_{\lambda} := \frac{1}{|B_k|} \sum_{j \in B_k} c_{\lambda}^{(j)},$$

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

$$\Sigma_{\lambda, \lambda'} = N \left( \delta_{\lambda, \lambda'} - \frac{1}{|B_k|} \right),$$

$$\left( \delta_{\lambda, \lambda'} \frac{c_{\lambda}}{N} - (1 - \delta_{\lambda, \lambda'}) \frac{c_{\lambda}}{N} \right),$$

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 $\omega$, Theorem 3 suggests that an eigendecomposition of $\Sigma$ 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 $H = (V, E)$ into a weighted graph $G$ by expanding cliques over each hyperedge. Next, $G$ 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 $G := \{G_1, G_2, ..., G_k\}$. The partitioning terminates whenever the second-smallest eigenvalue of the symmetric Laplacian matrix $\lambda_2$ exceeds a threshold value $\lambda_2^{\max}$. $\lambda_2^{\max}$ is dataset independent and thus fixed in our implementation. Finally, each subgraph $G_i$ is then converted into a hypergraph $H_i = (V_i, E_i)$ such that the vertex set $V_i$ of the hypergraph is initialised to be the vertex set of $G_i$. The edge set $E_i$ is then constructed by adding all hyperedges $e \in \hat{E}$ whose strict majority of element vertices appear in $V_i$, i.e. $E_i := \{e \in \hat{E} \mid |e \cap V_i| > |e|/2\}$. As a consequence, no nodes or edges are lost during clustering. HClustering returns the set of sub-hypergraphs $\{H_1, H_2, ..., H_k\}$. After partitioning, we run the rest of the pipeline with $L$ set to the diameter of each $H_i$.

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

### 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 $\varepsilon$-boundedness for all path signatures, we only run enough walks to guarantee $\varepsilon$-boundedness for the top $k$ most common path signatures.

---

**Algorithm 2:** PRISM

1. **Input:** $H$, the hypergraph representation of the input relational database
2. **Output:** Path-symmetric sets (abstract concepts $C$) of nodes w.r.t. each $v_i$ in $H$
3. **Parameters:** $\varepsilon, \alpha$
4. $H_1, \ldots, H_K := \text{HClustering}(H)$; // Sec. 4.2
5. Let $V_k$ denote the set of nodes in $H_k$
6. for $1 \leq k \leq K$
7. set $L$ to the diameter of $H_k$
8. compute $\varepsilon$-optimal $N$ on $H_k$ under $L$; // Th. 1
9. for each node $v_i$ in $H_k$
10. for each $v_j \neq v_i$ in $H_k$ compute $\hat{p}^{L,N}_{i,j}$ and $\hat{h}^{L,N}_{i,j}$; // Sec. 2
11. partition $V_k$ into distance-symmetric sets $\{A_1, A_2, ..., A_M\}$ using $\alpha$-significance; // Th. 2, Sec. 2
12. for $1 \leq m \leq M$
13. $C_m := \text{PathSymmetricClustering}(A_m, \alpha)$; // Th. 3, Sec. 4
14. return all $C_m$'s
We now illustrate the entire PRISM pipeline through an extended example, shown in Fig 3. In this example, describing a physics and history department in a university, we consider the hypergraph representation of a dataset. Nodes \( P_i \) are entities of type person, while \( B_i \) 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 \( \{P_4, P_5, P_{10}, P_{11}\} \) 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 \( P_8 \) and \( B_4 \) 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 \( B_1 \), we show examples of the distance-symmetric and path-symmetric clusters that we obtain in (ii) and (iii), respectively. Note how \( \{P_1, P_2, P_3\} \) in (ii) is partitioned into \( \{P_1, P_2\} \) and \( \{P_3\} \) 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 \( \varepsilon \) is

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

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 this example, 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 \( \varepsilon \)-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 \( \varepsilon \) gives an upper bound of \( N = 1.6 \times 10^3 \).

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...
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 \((O(n) vs O(ln n))\) 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: UWCSE (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 UWCSE 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 \texttt{STARRINGIN}(movie,person), \texttt{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 UWCSE 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

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

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.
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; Roktaeschel 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 pre-specify 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 motif mining 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 $O(n \ln n)$ 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-symmetric 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


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).


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.


