Summarizing Significant Subgraphs by Probabilistic Logic Programming

Elena Bellodi[1*] Ken Satoh[2] Mahito Sugiyama[2,3]

[1]Department of Engineering, University of Ferrara, Ferrara, Italy
[2]National Institute of Informatics, Tokyo, Japan
[3]JST, PRESTO
[1]elena.bellodi@unife.it
[2]{mahito,ksatoh}@nii.ac.jp

February 28, 2019

Abstract

Although recent advances of *significant subgraph mining* enable us to find subgraphs that are statistically significantly associated with the class variable from graph databases, it is challenging to interpret the resulting subgraphs due to their massive number and their propositional representation. Here we represent graphs by probabilistic logic programming and solve the problem of summarizing significant subgraphs by *structure learning of probabilistic logic programs*. Learning probabilistic logical models leads to a much more interpretable, expressive and succinct representation of significant subgraphs. We empirically demonstrate that our approach can effectively summarize significant subgraphs with keeping high accuracy.

Keywords Subgraph summarization Significant subgraph mining Statistical significance Probabilistic logic programming Logic programs with annotated disjunctions

*Corresponding author: Via Saragat 1 - Blocco A, Dipartimento di Ingegneria, 44122 Ferrara, Italy. Tel. +39 0532974882

1 Introduction

Pattern mining [1] is the process of finding multiplicative combinations of features (variables), or *patterns*, from a dataset, which has been actively studied as one of the central topics of data mining [42]. Various types of patterns have been used in applications. Examples include *itemsets* [13], which are combinations of binary features originally used in market basket analysis to find frequently co-purchased items, and *sequences* [24], used in DNA sequence analysis and customer behavior analysis. In this paper we focus on *subgraph mining* [14, 40], whose task is to find frequently occurring subgraphs from a collection of graphs, which is often called a graph database. Since a *graph* is a fundamental data structure and a wide range of graph-structured data is available, such as chemical compounds in PubChem [7] and protein structures in PDB [6], subgraph mining has been studied as an important branch of pattern mining to analyze graph-base data.

As an extension of the original subgraph mining problem, *significant* (*discriminative*) *subgraph mining* [19, 33] is recently attracting a considerable attention, which tries to find subgraphs enriched in one class relative to another class. For example, in drug discovery, each chemical compound is modeled as a graph and a collection of graphs is divided into two classes, case and control, and one can find subgraphs that are enriched in the case group while not in the control group. Significant subgraph mining offers to find all subgraphs that are *statistically significantly* associated with the class variable while correcting for multiple testing to ensure rigorous control of the FWER (family-wise error rate). This means that significant subgraph mining can rigorously control the

probability to detect one or more false positive subgraphs, which is indispensable in drug discovery and other scientific fields such as biology and medicine.

However, the challenge of significant pattern mining is that it often produces millions of significant subgraphs in a propositional representation, resulting in hard interpretability of the obtained subgraphs. How to summarize such massive amount of significant subgraphs in a principled way is still an open problem. Although pattern compression has been studied [1, Chapter 8], with the application of the MDL (Minimum Description Length) principle [38] to find representative patterns, or ILP based approaches have been used [11], none of the existing methods has been successfully applied to significant subgraph mining.

Our goal in this paper is to summarize significant subgraphs using Probabilistic Logic Programming to achieve better interpretability of massive subgraphs.

Probabilistic Logic Programming (PLP) is gaining popularity due to its ability to represent relational domains with many entities connected by complex and uncertain relationships. First-order logic is a powerful language to represent complex relational information, thanks to its intrinsic expressivity, while probability is the standard way to represent uncertainty in knowledge. One of the most fertile approaches to PLP is the distribution semantics [30], that is at the basis of several languages such as the Independent Choice Logic [25], PRISM [31], Logic Programs with Annotated Disjunctions (LPADs) [37] and ProbLog [10]. Various algorithms for learning parameters and structure of probabilistic logic programs written in these languages have been proposed, such as PRISM [32],

LFI-ProbLog [12] and EMBLEM [4] (for parameter learning), Sem-CP-Logic [20], CLP(BN) [8] and SLIPCOVER [5] (for structure learning).

In this paper, the key to mine a compact general representation of a collection of significant subgraphs is to use the state-of-the-art structure learning algorithm SLIPCOVER for probabilistic logic programs, which enables us to *encode a set of significant subgraphs as a probabilistic logical model written in the language of LPADs*. The advantages of building such a symbolic model are:

1. storing a logic program encoding significant subgraphs is significantly cheaper than storing all subgraphs, which are often too large in size and several in number; a single logical rule can describe many significant subgraphs at once;

2. a first-order logic-based representation of a collection of subgraphs is declarative and comprehensible by humans, and much more expressive than a propositional representation;

3. probability allows the management of uncertainty in complex application domains (such as the biological ones).

This paper provides the first application of Probabilistic Logic Programming to the problem of significant subgraph mining from standard biochemical datasets. The effectiveness of the approach, that we call LIPS for "Learning sIgnificant Plp Subgraphs", is empirically verified by showing that the precision and recall of the LPADs learnt by SLIPCOVER are better than those of a baseline of probabilistic logic programs with fixed probabilities. Since our objective is not

classifying graphs for predictive analysis but finding interpretable summarized representations of significant subgraphs for descriptive analysis, existing graph classification approaches (e.g. [15]) cannot be applied to our task.

The paper is organized as follows. Section 2 provides the necessary background about probabilistic logic programming and subgraph mining. Section 3 introduces the proposed approach based on PLP. Section 4 experimentally evaluates the method. Section 5 presents related work and Section 6 concludes the paper.

2 Background

2.1 Probabilistic Logic Programming

We assume that the reader is familiar with basic notions of First-Order Logic (FOL).

In this paper we rely on Probabilistic Logic Programming under the distribution semantics [30] for representing uncertain relational information. We consider Logic Programs with Annotated Disjunctions (LPADs) for their general syntax and we do not allow function symbols; for the treatment of function symbols see [29]. LPADs [37] allow to encode "alternatives" in the head of clauses in the form of a disjunction, in which each atom is annotated with a probability.

These programs consist of a finite set of annotated disjunctive clauses C_i of the form:

$$h_{i1} : \Pi_{i1}; \ldots; h_{in_i} : \Pi_{in_i} \ :- \ b_{i1}, \ldots, b_{im_i}.$$

Here, $b_{i1}, \ldots, b_{im_i}$ are logical literals which form the *body* of C_i, denoted by

$body(C_i)$, while $h_{i1}, \ldots h_{in_i}$ are logical atoms and $\{\Pi_{i1}, \ldots, \Pi_{in_i}\}$ are real numbers in the interval $[0, 1]$ such that $\sum_{k=1}^{n_i} \Pi_{ik} \leq 1$. Note that if $n_i = 1$ and $\Pi_{i1} = 1$ the clause corresponds to a non-disjunctive clause. Otherwise, if $\sum_{k=1}^{n_i} \Pi_{ik} < 1$, the head of the annotated disjunctive clause implicitly contains an extra atom *null* that does not appear in the body of any clause and whose annotation is $1 - \sum_{k=1}^{n_i} \Pi_{ik}$. The grounding of an LPAD $\mathcal{L}$ is denoted by $ground(\mathcal{L})$.

An *atomic choice* is a triple (C_i, θ_j, k) where $C_i \in \mathcal{L}$, θ_j is a substitution that grounds C_i and $k \in \{1, \ldots, n_i\}$ identifies a head atom of C_i. In other words, it represents the selection of the k-th atom from the head of the ground clause $C_i\theta_j$. It corresponds to an assignment $X_{ij} = k$, where X_{ij} is a multi-valued random variable which corresponds to $C_i\theta_j$. A set of atomic choices κ is *consistent* if only one head is selected from a ground clause. In this case it is called a *composite choice*. The *probability* $P(\kappa)$ *of a composite choice* κ is computed by multiplying the probabilities of the individual atomic choices, i.e. $P(\kappa) = \prod_{(C_i,\theta_j,k)\in\kappa} \Pi_{ik}$. A *selection* σ is a composite choice that, for each clause $C_i\theta_j$ in $ground(\mathcal{L})$, contains an atomic choice (C_i, θ_j, k). It identifies a *world* w_σ of $\mathcal{L}$, i.e., a normal logic program defined as $w_\sigma = \{(h_{ik} \leftarrow body(C_i))\theta_j | (C_i, \theta_j, k) \in \sigma\}$. Since selections are composite choices, the probability of the worlds is $P(w_\sigma) = P(\sigma)$. We denote by $S_\mathcal{L}$ the set of all selections and by $W_\mathcal{L}$ the set of all worlds of a program $\mathcal{L}$.

We consider only *sound* LPADs, where each possible world has a total well-founded model, so $w_\sigma \models Q$ means that the query Q is true in the well-founded model of the program w_σ. The probability of a query Q given a world w is

$P(Q|w) = 1$ if $w \models Q$ and 0 otherwise. The probability of Q is then:

$$P(Q) = \sum_{w \in W_{\mathcal{L}}} P(Q, w) = \sum_{w \in W_{\mathcal{L}}} P(Q|w)P(w) = \sum_{w \in W_{\mathcal{L}}: w \models Q} P(w) \qquad (1)$$

Example 1 *The following LPAD $\mathcal{L}$ encodes the development of an epidemic or pandemic:*

$$C_1 = epidemic : 0.6;\ pandemic : 0.3\ :-\ \ flu(X), cold.$$

$$C_2 = cold : 0.7.$$

$$C_3 = flu(david).$$

$$C_4 = flu(robert).$$

This LPAD models the fact that if somebody has the flu and the climate is cold the possibility that an epidemic arises has probability 0.6 *to be true, that a pandemic arises has probability* 0.3 *or that no event happens (the implicit atom null) has probability* 0.1. *There is uncertainty about the climate, it may be cold with a probability of* 0.7 *but surely David and Robert have the flu.*

Clause C_1 has two groundings, $C_1\theta_1$ with $\theta_1 = \{X/david\}$ and $C_1\theta_2$ with $\theta_2 = \{X/robert\}$ so there are two random variables X_{11} and X_{12}. $\mathcal{L}$ has 18 possible worlds, the query $Q = epidemic$ is true in 5 of them and its probability is obtained as $P(epidemic) = 0.6 \cdot 0.6 \cdot 0.7 + 0.6 \cdot 0.3 \cdot 0.7 + 0.6 \cdot 0.1 \cdot 0.7 + 0.3 \cdot 0.6 \cdot 0.7 + 0.1 \cdot 0.6 \cdot 0.7 = 0.588$.

The semantics associates one random variable with every grounding of a clause. In some domains, this may result in too many random variables, so we may introduce an approximation at the level of the instantiations, at the expenses of

the accuracy in modeling the domain. A typical compromise is to consider the grounding of variables in the head only: in this way, a ground atom entailed by two separate ground instances of a clause is assigned the same probability, all other things being equal, of a ground atom entailed by a single ground clause, while in the standard semantics the first would have a larger probability as more evidence is available for its entailment. In the approximate semantics clause C_1 of Example 1 is associated to a single random variable X_1. In this case $\mathcal{L}$ has 6 instances, the query epidemic is true in 1 of them and its probability is $P(epidemic) = 0.6 \cdot 0.7 = 0.42$.

An efficient technique for computing the probability of a query consists of building a Binary Decision Diagram (BDD), representing the disjunction of its explanations, and performing inference over it. Inference can be performed with a dynamic programming algorithm that is linear in the size of the BDD [10]. Algorithms that adopt such an approach for inference include [26, 27, 28]. BDDs can be built in practice by highly efficient software packages such as CUDD[1].

2.2 Significant Subgraph Mining

In significant subgraph mining, each graph is defined as a triple $G = (V, E, \phi)$ composed of the vertex set V, the edge set $E \subseteq V \times V$, and the label mapping $\phi : V \cup E \to \Sigma$ with the range Σ of vertex and edge labels. A graph $H = (V', E', \phi')$ is a *subgraph* of G, denoted by $H \sqsubseteq G$, if $V' \subseteq V$, $E' \subseteq (V' \times V') \cap E$, and $\phi'(A) = \phi(A)$ for all $A \in V' \cup E'$ are satisfied. Given two collections of graphs $\mathcal{G}$ and $\mathcal{G}'$ with $|\mathcal{G}| = n$ and $|\mathcal{G}'| = n'$, we assume $n \leq n'$ without loss of generality.

[1] Available at `http://vlsi.colorado.edu/~fabio/CUDD/`

$\mathcal{G}$ and $\mathcal{G}'$ represent two different classes of graphs that we want to distinguish. In Figure 1, we have four graphs in $\mathcal{G}$ and also four graphs in $\mathcal{G}'$, and colors of vertices and line types of edges denote their labels.

For each subgraph $H \sqsubseteq G$ with $G \in \mathcal{G} \cup \mathcal{G}'$, we test the *statistical association* between the occurrence of H and the class membership, where the null hypothesis is that the occurrence of the subgraph H is *independent* of the class membership of G. More precisely, we measure the statistical association between two binary random variables: the indicator vector of the class membership of graph G and the occurrence/absence of the subgraph H in each graph G in the $\mathcal{G}$ and $\mathcal{G}'$ databases.

Let x and x' be the frequencies of H in $\mathcal{G}$ and $\mathcal{G}'$, respectively. That is,

$$x = |\{G \in \mathcal{G} \mid H \sqsubseteq G\}|, \quad x' = |\{G \in \mathcal{G}' \mid H \sqsubseteq G\}|.$$

Then the occurrence of H can be represented as the following 2×2 contingency table:

	Occurrences	Non-occurrences	Total
$\mathcal{G}$	x	$n - x$	n
$\mathcal{G}'$	x'	$n' - x'$	n'
Total	$x + x'$	$(n - x) + (n' - x')$	$n + n'$

For example, for the subgraph shown in Figure 1, $x = 4$, $x' = 0$, and $n = n' = 4$. The association between two binary random variables is measured by *Fisher's exact test* as the *p-value*, which is the probability of false positives assuming that the null hypothesis is true, that is, occurrences of the subgraph and classes are statistically independent. The false positive occurs if we reject the null hypothesis while it is true. We say that a subgraph H is *statistically significant*

if its p-value is smaller than a predetermined significance level α.

The technique of significant subgraph mining proposed in [19, 33] finds all subgraphs that are *statistically significantly* associated with the class variable, that is, the above null hypothesis is rejected, while correcting for multiple testing to ensure rigorous control of the FWER (family-wise error rate). The FWER is the probability that at least one subgraph is a false positive in the set of all subgraphs (hypotheses). Since the FWER approaches one even if the false positive rate is controlled under the significance level α for each subgraph, the significant subgraph mining technique performs multiple testing correction by decreasing the significance level α so that the condition FWER $\leq \alpha$ is guaranteed.

3 The Proposed Method: LIPS

We provide here a detailed description of our proposed method, called LIPS (Learning sIgnificant Plp Subgraphs), in three subsequent steps, which summarize a given set of significant subgraphs as a probabilistic logic program.

3.1 First-order Logic Representation of Subgraphs

Here we introduce the first step, which provides a FOL representation of a given set of subgraphs.

Interestingly, the significant subgraph mining technique [33] always produces the set of *testable subgraphs* as an intermediate result for the FWER control. Mathematically, a testable subgraph is defined in the following. The tight lower

bound $\psi(H)$ of the p-value for a subgraph H is given as

$$\psi(H) = \begin{cases} \binom{n}{x} \Big/ \binom{n+n'}{x} & \text{if } 0 \leq x + x' \leq n, \\ 1 \Big/ \binom{n+n'}{x} & \text{otherwise,} \end{cases}$$

This was firstly considered in [35] and used in [33]. Assume that the set of subgraphs is sorted in increasing order according to the lower bound ψ, resulting in the sequence

$$\psi(H_1) \leq \psi(H_2) \leq \psi(H_3) \leq \dots.$$

Let k be the natural number such that

$$k \cdot \psi(H_k) < \alpha \quad \text{and} \quad (k+1) \cdot \psi(H_{k+1}) \geq \alpha.$$

The subgraphs $H_1, H_2, \dots, H_k$ are defined to be *testable* subgraphs and each testable subgraph H_i, $i \in \{1, 2, \dots, k\}$ is statistically significant if its actual p-value is smaller than α/k [34].

Given two sets of graphs $\mathcal{G}$ and $\mathcal{G}'$, let $\mathcal{T}$ and $\mathcal{S}$ be the sets of testable subgraphs and significant subgraphs respectively, produced by significant subgraph mining. Since we always have the relationship $\mathcal{S} \subseteq \mathcal{T}$, we formulate the problem of summarization as discriminating positive instances (significant subgraphs) $\mathcal{S}$ from negative instances (testable but non-significant subgraphs) $\mathcal{T} \setminus \mathcal{S}$.

Our method takes as input a set of testable subgraphs, which includes positive and negative instances, and transforms them into a set of corresponding logical interpretations (sets of ground facts). Let $G = (V, E, \phi) \in \mathcal{T}$ be a graph such that $V = \{v_1, v_2, \dots, v_m\}$, $E = \{\{v_{i_1}, v_{j_1}\}, \{v_{i_2}, v_{j_2}\}, \dots, \{v_{i_l}, v_{j_l}\}\}$, and

$\phi : V \cup E \to \Sigma$ assigns labels to vertices and edges with the range Σ. Its logical representation is defined as:

$$\texttt{node}(v_1, \phi(v_1)).$$

$$\texttt{node}(v_2, \phi(v_2)).$$

$$\ldots$$

$$\texttt{node}(v_m, \phi(v_m)).$$

$$\texttt{edge}(v_{i_1}, v_{j_1}, \phi(\{v_{i_1}, v_{j_1}\})).$$

$$\texttt{edge}(v_{i_2}, v_{j_2}, \phi(\{v_{i_2}, v_{j_2}\})).$$

$$\ldots$$

$$\texttt{edge}(v_{i_l}, v_{j_l}, \phi(\{v_{i_l}, v_{j_l}\})).$$

In the above representation, `node/2` describes a node by means of the node's id and the node's label (`node(id,label)`), while `edge/3` describes an edge by means of the two nodes linked by the edge and the edge's label (`edge(id1,id2,label)`).

For example, if $G = (V, E, \phi)$ is given as $V = \{0, 1, 2, 3, 4\}$, $E = \{\{0, 1\}, \{1, 2\}, \{2, 3\}, \{3, 4\}\}$, node labels are given as $\phi(0) = 3$, $\phi(1) = 3$, $\phi(2) = 3$, $\phi(3) = 3$, $\phi(4) = 6$, and edge labels as $\phi(\{0, 1\}) = 47$, $\phi(\{1, 2\}) = 47$, $\phi(\{2, 3\}) = 47$, $\phi(\{3, 4\}) = 50$, the resulting logical interpretation for G is:

```
node(0,3).
node(1,3).
node(2,3).
node(3,3).
```

```
node(4,6).

edge(0,1,47).

edge(1,2,47).

edge(2,3,47).

edge(3,4,50).
```

Finally, an additional predicate `active/0` is used to discriminate between positive and negative instances: a fact `active.` or `neg(active).` will be respectively added to each logical interpretation. It is clear that each graph G and its logical representation are in a one-to-one relationship.

3.2 Learning a representation of Significant Subgraphs by Probabilistic Logic Programming

Given the set of significant subgraphs $\mathcal{S}$ and the set of testable subgraphs $\mathcal{T}$, LIPS learns rules that can probabilistically discriminate significant subgraphs $\mathcal{S}$ from non-significant subgraphs $\mathcal{T} \setminus \mathcal{S}$ by means of a *compact* probabilistic logic program (in particular, LPAD). In other words, the problem of significant subgraph mining is converted in a structure learning problem of probabilistic logic clauses imposing constraints on the labels and connection structure of the original subgraphs. We illustrate an overview of LIPS in Figure 2. To learn LPADs we employ the SLIPCOVER algorithm, which is a state-of-the-art learning algorithm and has been successfully applied in various relational domains [5]. The reason for this choice is the fact that graph-structured datasets characterized by links between nodes are inherently relational. We briefly summarize SLIPCOVER in the following and give a detailed execution example in the next subsection for a

better understanding of its behavior.

Input consists of a set of logical interpretations I, i.e. sets of ground facts as seen in subsection 3.1, corresponding to the testable subgraphs which need to be discriminated. The algorithm is targeted at discriminative learning, that is the user has to indicate which predicate(s) of the domain is/are *target*, the one(s) for which we are interested in good predictions. The interpretations must contain also negative facts for target predicates. The ground atoms for the target predicates represent the positive and negative examples (*queries*) for which Binary Decision Diagrams will be built, encoding the disjunction of their explanations.

SLIPCOVER is built upon a two-phase search strategy: (1) a beam search in the space of clauses in order to find a set of promising clauses and (2) a greedy search in the space of theories. In the first phase the beam for each target predicate is initialized with a number of bottom clauses built as in Progol [21], which are repeatedly refined according to a "language bias". The second phase starts from an empty theory which is iteratively extended with one target clause at a time from those generated in the previous phase. If the log-likelihood (LL) of the new theory does not increase, SLIPCOVER discards the clause, otherwise it adds it to the current theory.

BDDs are employed to efficiently perform the parameter learning phase of the LPAD, i.e. to compute the optimum probabilities for the clauses' heads. This is done by the algorithm EMBLEM [4], based on an Expectation Maximization (EM) cycle. Both parameter and structure learning use the log-likelihood of the

data as the guiding heuristic to find the best parameters and the best theory. This guarantees that the final LPAD returned by SLIPCOVER locally maximizes the (log-)likelihood with respect to the set of positive and negative examples (subgraphs) for the target predicate(s). LIPS is shown in Alg. 1, while a simplified version of SLIPCOVER (relevant for the understanding of the method) in Alg. 2.

Algorithm 1 Function LIPS

```
1: function LIPS(G, G', target)
2:     (T, S) =MineSignificantSubgraphs(G, G')                                  ▷
3:     L =Slipcover(I, target)          ▷ Input interpretations I = T ∪ S; L: learned LPAD
4:     return L
5: end function
```

Algorithm 2 Function SLIPCOVER

```
1: function SLIPCOVER(I, target)
2:     IBs =InitialBeams(I, target)    ▷ Beam search returns a set of beams, one for each target predicate
3:     TC ← []                          ▷ TC: list of promising clauses with target predicate in the head
4:     for all Beam ∈ IBs do
5:         Steps ← 1
6:         NewBeam ← []
7:         repeat
8:             while Beam is not empty do
9:                 Remove the first BC from Beam                              ▷ BC: Bottom Clause
10:                Refs ←ClauseRefinements(BC)          ▷ Find all refinements Refs of BC
11:                for all Cl ∈ Refs do                                       ▷ Cl:refined clause
12:                    (LL', Cl') ←EMBLEM(Cl)   ▷ Parameter learning: updates Cl's parameters and computes the log-likelihood LL' of the new clause Cl'
13:                    NewBeam ←Insert(Cl', LL', NewBeam)
14:                    TC ←Insert(Cl', LL', TC)
15:                end for
16:            end while
17:            Beam ← NewBeam
18:            Steps ← Steps + 1
19:        until Steps > NI                                  ▷ NI: max number of iterations
20:    end for
21:    L ← ∅, LL_L ← −∞                                  ▷ Greedy search: initial LPAD empty
22:    repeat
23:        Remove the first couple (Cl, LL) from TC
24:        (LL', L') ←EMBLEM(L ∪ {Cl})
25:        if LL' > LL_L then
26:            L ← L', LL_L ← LL'
27:        end if
28:    until TC is empty
29:    return L
30: end function
```

3.3 Execution Example

The analyzed domains are graphs of chemical compounds, with nodes labeled according to the atom type (predicate `node/2`) and edges that represent the bonds (predicate `edge/3`). Significant subgraph mining produces testable subgraphs including significant ones from a given set of graphs, and our method summarizes them. The target predicate is `active/0`, where significant subgraphs are encoded as active and testable but not significant subgraphs are encoded as non-active. A language bias is given to specify modeh/modeb declarations for building the bottom clauses and their refinements. Such declarations are templates for literals in the head or body of a clause [21]. For all the considered domains the language bias has been defined as follows.

```
output(active/0).
input(node/2).
input(edge/3).
modeh(1,active).
modeb(*,node(-node,-label)).
modeb(*,edge(+node,-node,-label)).
modeb(*,edge(-node,+node,-label)).
modeb(*,edge(+node,+node,-label)).
```

output indicates the target predicate, while *input* the other predicates of the domain, together with their arity. The *modeh* predicate indicates that at most 1 occurrence of `active` must be used in the clauses' heads. The *modeb* predicates

indicate that any number of occurrences (*) of the `node` and `edge` predicates can be used to build the clauses' bodies. Each different graph is represented by a different input FOL interpretation.

The output of SLIPCOVER consists of LPAD theories whose clauses predict the target predicate with a given probability Π_{i1}, as a function of a specific configuration of typed nodes connected by typed edges. Π_{i1} indicates the probability of the first head atom (the only one, since we specified modeh(*1*,active)) for each clause i. An example of LPAD (composed of a single clause) returned by the algorithm on one of the domains (MUTAG_5) is:

```
active:0.0411023 :- node(A,B),edge(A,C,D),node(C,B),edge(C,E,D),
                    edge(C,F,D),node(E,B),node(F,B).
```

The clause states that a subgraph where some nodes A, C, E, F are of type B, and there is a direct edge between nodes A and C, C and E, C and F of type D is testable significant (active), which is illustrated in Figure 3. The figure shows how several subgraphs can be compactly represented by a single clause. The value $\Pi_{11} = 0.0411023$ represents the probability with which each grounding satisfying the clause body (i.e., an explanation) is active. This means that if there are more explanations, the probability of describing a significant subgraph pattern increases, according to the formula of the probability of the union of compatible events ($P(A \cup B) = P(A) + P(B) - P(A \cap B)$ for two events A and B). In this example, by asking the probability of two positive (active) interpretations in the test set we get $P = 0.2226$ for both, while for two negative interpretations (labelled as neg(active)) in the test set we get probabilities $P = 0.0411$ and

0.0805, which are much smaller than 0.2226. The clause has assigned larger probabilities to the positive instances (significant subgraphs).

4 Experimental Validation

In this section we empirically evaluate the performance of LIPS on standard graph datasets. These datasets include: MUTAG, NCI1 and NCI109[2]. The MUTAG dataset consists of graphs representing 188 chemical compounds, and aims to predict whether each compound shows mutagenicity. The NCI1 and NCI109 datasets consist of graphs representing two balanced datasets of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines respectively. For each dataset, we first enumerated the set of testable subgraphs by the significant subgraph mining algorithm technique [33][3]. We created different versions of testable subgraphs from these datasets with a different maximum number $N \in \{3, 5, 10\}$ of subgraph nodes. In Table 1, we denote by "_N" following the dataset's name the maximum number N of nodes. For example, MUTAG_5 means that the number of nodes of every testable subgraph is less than or equals to 5. More information about the characteristics of the datasets is shown in Table 1.

For training and test we employed a k-fold cross validation, with k depending on the dataset (see below).

In order to verify the effectiveness of the approach, we tested the same LPADs

[2]These datasets are available at `https://www.bsse.ethz.ch/mlcb/research/machine-learning/graph-kernels/weisfeiler-lehman-graph-kernels.html`

[3]Implementation is available at `https://github.com/BorgwardtLab/significant-subgraph-mining`

Table 1: Characteristics of graph datasets. #Pos ex denotes the number of positive examples (significant subgraphs), while #Neg ex denotes the number of negative examples (testable but non-significant subgraphs). The number of ground facts (column 5) does not include facts for the target predicate. #Graphs is the total number of subgraphs in each dataset.

Datasets	#Pos ex	#Neg ex	#Ground facts	#Graphs
MUTAG_5	8	221	2121	188
MUTAG_10	1054	2277	67065	188
NCI1_3	121	254	2338	4110
NCI1_10	83,300	133,606	4,620,649	4110
NCI109_3	118	250	2293	4127

learnt by SLIPCOVER where parameters were replaced with fixed and randomly chosen values, i.e. we tested the same clauses with non-optimized parameters on the datasets. All experiments were performed on GNU/Linux machines with an Intel Xeon Haswell E5-2630 v3 (2.40GHz) CPU with 128 GB RAM.

4.1 Learning

In order to reach a compromise between accuracy (performance) and learning time, we tuned the following SLIPCOVER settings: the type of semantics (standard or simplified), the limit on the number of different solutions retrieved when computing the probability of a query, the maximum number of theory search iterations, the maximum number of clause search iterations, the size of the beam, the maximum number of variables in a rule.

For MUTAG_10 and NCI1_10, we employed a 10-fold cross-validation due to the large number of positive and negative examples. For NCI1_3 and NCI109_3 we employed a 5-fold cross-validation. For MUTAG_5 we employed a 4-fold cross-validation due to the presence of only 8 positive examples. This information is reported in Table 2.

As an example, we report in the following the output of the learning algorithm on one fold of the MUTAG_10 dataset, a LPAD composed of 3 probabilistic logical clauses:

```
active:0.00245023 :- node(A,B),edge(C,A,D),node(E,B),edge(A,F,D),
                     node(G,B),edge(A,E,D),edge(G,H,D),edge(H,I,D),
                     node(H,B),edge(I,J,D),node(J,B),node(F,B).

active:0.00372679 :- node(A,B),edge(C,A,D),node(E,B),edge(A,F,D),
                     node(G,B),edge(A,E,D),edge(G,H,D),edge(H,I,D),
                     node(C,B),edge(I,J,D),node(H,B),node(I,B),
                     node(J,B),node(F,B).

active:0.00488592 :- node(A,B),edge(C,A,D),node(E,B),edge(A,F,D),
                     node(G,B),edge(A,E,D),edge(G,H,D),edge(H,I,D),
                     node(C,B),node(H,B),node(I,B).
```

4.2 Test

We computed the Area Under the Precision-Recall Curve (AUC-PR) for the probabilistic logic programs learned by SLIPCOVER and for the programs with the same structure but fixed parameters ("baseline").

The Precision-Recall Curves have been obtained by collecting the testing examples, together with the probabilities assigned to them in testing by the LPADs, in a single set and then building the curves with the method reported

in [9].

Table 2 shows the comparison between LIPS (in terms of area under the PR curve, SLIPCOVER learning time, number of LPAD clauses and number of body literals per clause, all averaged over the folds) and the baseline (in terms of area under the PR curve averaged over the folds). The comparison with a baseline of LPADs with fixed parameters demonstrates that LIPS successfully learned a more accurate summarization for significant subgraphs in a short time in all cases except for the NCI1_10 dataset. Even in the case of NCI1_10, composed of thousands of examples, the obtained AUCPR is larger than the baseline.

As for the number of clauses and body literals of the learned LPADs, Table 2 shows that we can get a very concise description of significant subgraphs, with less than 8 clauses in the analyzed domains, and with short clauses in most cases.

The possibility of tackling the problem of summarizing significant subgraphs by means of the SLIPCOVER system comes from the relational nature of the tested domains and from the discriminative learning setting of the algorithm; this property has been exploited to distinguish significant from non-significant subgraphs by means of a *target* predicate (active) in the first-order logic representation.

5 Related Work

Significant pattern mining have been firstly achieved in [35] in the context of itemset mining [2, 3] using the Tarone's testability trick [34] and further developed in [19, 36] by considering Westfall-Young permutation test [39]. Such

Table 2: Results of the experiments comparing LIPS with a baseline of LPADs with fixed parameters for each dataset, in terms of average Area Under the PR Curve (AUC-PR), SLIPCOVER average execution time (in seconds), average number of LPAD clauses and average number of body literals per clause. Column "Folds" specifies the number of folds used for cross-validation.

Dataset	Folds	**Baseline**	**LIPS**			
		AUC-PR	AUC-PR	Time(s)	Clauses	Literals
MUTAG_5	4	0.66	0.82	156.15	1	7.25
MUTAG_10	10	0.61	0.73	0.38	3.9	12.28
NCI1_3	5	0.40	0.41	33.86	5.6	3
NCI1_10	10	0.38	0.48	58889.45	2.3	2
NCI109_3	5	0.34	0.45	35.67	7.2	3.56

methods can find all statistically significant patterns from databases while rigorously controlling the FWER. Recently, significant pattern mining has been applied to various data including interval data [17] and datasets with categorical covariates [23]. The software library of significant pattern mining is available [18] and an efficient parallelized implementation is also available [41].

Sugiyama et al. [33] applied significant pattern mining to graph structured data using subgraph mining algorithms [22, 40] and established *significant subgraph mining*. The technique can find all statistically significant subgraphs while controlling the FWER. However, since significant subgraph mining tends to generate a huge number of significant subgraphs, how to summarize such subgraphs is a challenging task for further analysis in applications. This paper has addressed the problem using probabilistic logic programming for the first time. The ProbLog language was employed in the context of local query mining from probabilistic biological databases [16], but with a different goal than significant subgraph mining.

6 Conclusions

We proposed the first method to find a compact general representation of a collection of significant subgraphs in the form of (probabilistic) logic programs. This was achieved through the application of a structure learning algorithm for probabilistic logic programs, SLIPCOVER, to standard graph-based datasets. The key idea is to formulate the problem of summarization of significant subgraphs as classification of testable and significant subgraphs to allow the application of learning algorithms of probabilistic logic programs. Experiments show that we can massively compress the set of significant subgraphs with reasonably high precision and recall. Since significant pattern summarization is the problem of not only subgraph mining but the general setting of pattern mining including itemset mining and sequence mining, our approach combining significant pattern mining and probabilistic logic programming has an impact to a wider range of applications of significant pattern mining. For instance, the approach might be applied to many interesting applications in chemoinformatics, structural biology, and precision medicine. In the future, we plan to apply other structure learning algorithms of logic programs targeted to big knowledge bases in order to reduce the computational time.

Acknowledgments This work has been done in the context of a research agreement between the University of Ferrara (Italy) and the National Institute of Informatics (Tokyo, Japan). Mahito Sugiyama has been supported by JSPS KAKENHI Grant Numbers JP16K16115, JP16H02870, and JST, PRESTO

Grant Number JPMJPR1855, Japan. Elena Bellodi has been partially supported by the Italian National Group of Computing Science (GNCS-INDAM).

References

[1] C. C. Aggarwal and J. Han, editors. *Frequent Pattern Mining*. Springer, 2014.

[2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, 1994.

[4] Elena Bellodi and Fabrizio Riguzzi. Expectation Maximization over Binary Decision Diagrams for probabilistic logic programs. *Intelligent Data Analysis*, 17(2):343–363, 2013.

[5] Elena Bellodi and Fabrizio Riguzzi. Structure learning of probabilistic logic programs by searching the clause space. *Theory and Practice of Logic Programming*, 15(2):169–212, 2015.

[6] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

[7] E. E. Bolton, Y. Wang, P. A. Thiessen, and S. H. Bryant. PubChem: integrated platform of small molecules and biological activities. *Annual reports in computational chemistry*, 4:217–241, 2008.

[8] Vítor Santos Costa, David Page, Maleeha Qazi, and James Cussens. CLP(BN): constraint logic programming for probabilistic knowledge. *CoRR*, abs/1212.2519, 2012.

[9] J. Davis and M. Goadrich. The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 233–240, 2006.

[10] L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, volume 7, pages 2462–2467, 2007.

[11] Paul Finn, Stephen Muggleton, David Page, and Ashwin Srinivasan. Pharmacophore discovery using the inductive logic programming system progol. *Machine Learning*, 30(2):241–270, 1998.

[12] Bernd Gutmann, Ingo Thon, and Luc De Raedt. Learning the parameters of probabilistic logic programs from interpretations. In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, editors, *European Conference on Machine Learning and Knowledge Discovery in Databases*, volume 6911 of *LNCS*, pages 581–596. Springer, 2011.

[13] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 1–12, 2000.

[14] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Principles of Data Mining and Knowledge Discovery*, volume 1910 of *LNCS*, pages 13–23. Springer, 2000.

[15] N. Jin, C. Young, and W. Wang. Graph classification based on pattern co-occurrence. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 573–582, 2009.

[16] Angelika Kimmig and Luc De Raedt. Local query mining in a probabilistic Prolog. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, volume 2, pages 1095–1100, 2009.

[17] F. Llinares-López, D. G. Grimm, D. A. Bodenham, U. Gieraths, M. Sugiyama, B. Rowan, and K. M. Borgwardt. Genome-wide detection of intervals of genetic heterogeneity associated with complex traits. *Bioinformatics*, 31(12):i240–i249, 2015.

[18] F. Llinares-López, L. Papaxanthos, D. Roqueiro, D. Bodenham, and K. Borgwardt. CASMAP: detection of statistically significant combinations of SNPs in association mapping. *Bioinformatics*, 12 2018.

[19] F. Llinares-López, M. Sugiyama, L. Papaxanthos, and K. M. Borgwardt. Fast and memory-efficient significant pattern mining via permutation testing. In

Proceedings of the 21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 725–734, 2015.

[20] W. Meert, J. Struyf, and H. Blockeel. Learning ground CP-Logic theories by leveraging Bayesian network learning techniques. *Fundamenta Informaticae*, 89(1):131–160, 2008.

[21] Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[22] S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 647–652, 2004.

[23] L. Papaxanthos, F. Llinares-Lopez, D. Bodenham, and K. M. Borgwardt. Finding significant combinations of features in the presence of categorical covariates. In *Advances in Neural Information Processing Systems*, volume 29, pages 2271–2279, 2016.

[24] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M-.C. Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 17th International Conference on Data Engineering*, pages 215–224, 2001.

[25] David Poole. The Independent Choice Logic and beyond. In Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors, *Probabilistic Inductive Logic Programming*, volume 4911 of *LNCS*, pages 222–243. Springer Berlin Heidelberg, 2008.

[26] Fabrizio Riguzzi. Speeding up inference for probabilistic logic programs. *The Computer Journal*, 57(3):347–363, 2014.

[27] Fabrizio Riguzzi and Terrance Swift. Tabling and answer subsumption for reasoning on logic programs with annotated disjunctions. In *International Conference on Logic Programming*, volume 7 of *LIPIcs*, pages 162–171, 2010.

[28] Fabrizio Riguzzi and Terrance Swift. The PITA system: Tabling and answer subsumption for reasoning under uncertainty. *Theory and Practice of Logic Programming*, 11(4–5):433–449, 2011.

[29] Fabrizio Riguzzi and Terrance Swift. Well-definedness and efficient inference for probabilistic logic programming under the distribution semantics. *Theory and Practice of Logic Programming*, 13(2):279–302, 2013.

[30] Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming*, pages 715–729, 1995.

[31] Taisuke Sato. A glimpse of symbolic-statistical modeling by PRISM. *Journal of Intelligent Information Systems*, 31(2):161–176, 2008.

[32] Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.

[33] M. Sugiyama, F. Llinares-López, N. Kasenburg, and K. M. Borgwardt. Significant subgraph mining with multiple testing correction. In *Proceedings*

of the 2015 SIAM International Conference on Data Mining, pages 37–45, 2015.

[34] R. E. Tarone. A modified Bonferroni method for discrete data. *Biometrics*, 46(2):515–522, 1990.

[35] A. Terada, M. Okada-Hatakeyama, K. Tsuda, and J. Sese. Statistical significance of combinatorial regulations. *Proc. Natl. Acad. Sci. USA*, 110(32):12996–13001, 2013.

[36] A. Terada, K. Tsuda, and J. Sese. Fast Westfall-Young permutation procedure for combinatorial regulation discovery. In *2013 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 153–158, 2013.

[37] J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic Programs With Annotated Disjunctions. In *International Conference on Logic Programming*, volume 3131 of *LNCS*, pages 195–209. Springer, 2004.

[38] J. Vreeken, M. van Leeuwen, and A. Siebes. KRIMP: Mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[39] P. H. Westfall and S. S. Young. *Resampling-based multiple testing: Examples and methods for p-value adjustment*. John Wiley & Sons, 1993.

[40] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of 2002 IEEE International Conference on Data Mining*, pages 721–724, 2002.

[41] K. Yoshizoe, K. Tsuda, and A. Terada. MP-LAMP: parallel detection of statistically significant multi-loci markers on cloud platforms. *Bioinformatics*, 34(17):3047–3049, 04 2018.

[42] M. J. Zaki and W. Meira Jr. *Data Mining And Analysis*. Cambridge, 2016.

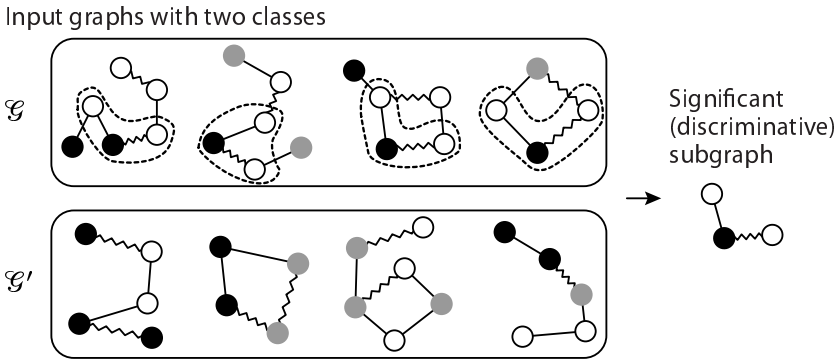

Figure 1: The problem setting of significant subgraph mining.

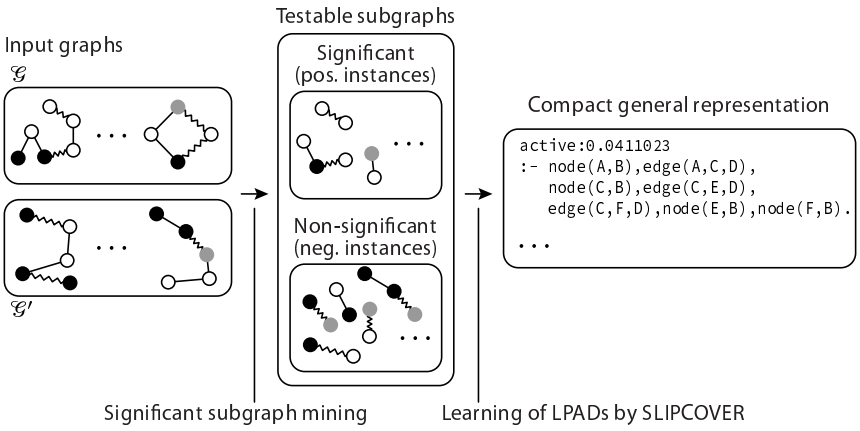

Figure 2: Overview of the proposed method LIPS.

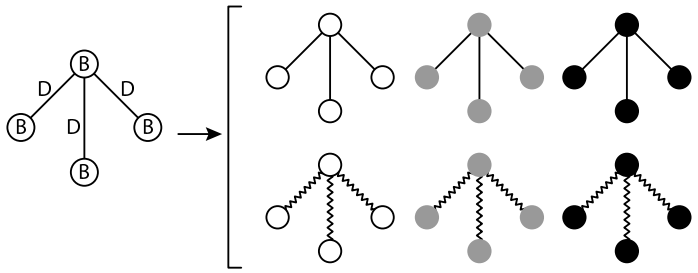

Figure 3: The left graph corresponds to `node(A,B), edge(A,C,D), node(C,B), edge(C,E,D), edge(C,F,D), node(E,B), node(F,B)`. If the domain contains three node labels (blue, green, red) and two edge labels (straight, zigzag) then there could be six possible subgraphs (instances).