Learning Graph Grammars

Aly El Gamal

ECE Department and Coordinated Science Laboratory

University of Illinois at Urbana-Champaign

Abstract

Discovery of patterns in a given graph - in the form of repeated subgraphs - is useful in finding a concise representation of the graph, as well as identifying objects of interest to the overlying application. In many application scenarios of practical significance, available background knowledge can lead to an extension of a subgraph discovery algorithm to infer general grammar rules of a language that describes a class of graphs, to which the input graph belongs. The goal of this work is three fold. We first consider the binary encoding scheme of directed graphs introduced in [1], and compute the description length of an input graph using this representation. We then propose a subgraph discovery algorithm, where each discovered subgraph reduces the total description length of an input graph. Finally, we propose a probabilistic framework for the inference of graph grammar rules that apply to other graphs that are similar but different from the input graph, where the measure of similarity can be set by the overlying application.

I. Introduction

Graphs provide a useful tool for expressing structured data as they can represent relations between different data items. Moreover, in directed graphs, causal relations can be represented. In this work, we study the problem of discovering patterns in structured data, through the discovery of subgraphs that occur more than once in a given graph. A subgraph discovery algorithm aids the overlying application in identifying objects of interest as well as finding a concise representation for the input graph. Given an input graph, more than one representation of the graph in terms of its repeated subgraphs can exist. In order to select one such representation, the authors in [1] suggest the use of the minimum description length principle introduced by Rissanen in [2]. To that end, they present a binary encoding scheme for directed graphs, and then a subgraph is selected if it minimizes the sum description lengths of the representation of the subgraph and the representation of the original graph in terms of the subgraph. In Section II, we provide a computation of the description length of a graph given the encoding scheme in [1], and use it as a guideline for our proposed subgraph discovery algorithm in Section III. The proposed algorithm saves only discovered subgraphs that reduces the total description length of the input graph.

Recognition of patterns in structured data can be aided by background and prior knowledge to infer more general rules that describe a class of structures containing the input samples. A famous example is that of acquiring the grammatical rules of a language by children, only through exemplary statements. The child can, from finitely many examples, with a probability that is rather high, deduce - with sufficient accuracy - the rules of the native language, that enables the generalization of the finite set of sampled sentences to an infinite one. In [3], the author comments

"This poverty of stimulus in the child language acquisition process motivated Chomsky to suggest that children operate with hypotheses about language which are constrained in some fashion.". Similarly, a learning algorithm uses prior knowledge to construct a set of hypotheses, and given input data, selects the hypothesis that best describes the data. According to Occam's razor, the simplest consistent hypothesis is the one that should be selected.

In [4], the authors introduce the SubdueGL algorithm for inferring grammatical rules of languages that describe classes of graphs. SubdueGL uses a subgraph discovery algorithm that considers the possibility of graph vertices with varying labels, and recursion rules that allow a graph to have an arbitrary number of repetitions of a given subgraph. In Section IV, we discuss application scenarios where SubdueGL infers the same grammar rules for graphs that are considered dissimilar by the application. We then suggest a probabilistic framework for the inference of grammatical rules of languages that describe classes of graphs. Hypotheses about the language can be embedded in the framework through prior probabilities of grammar rules. Furthermore, consistency of a selected hypothesis can be verified using the conditional probability of the input graph given the inferred grammar rule.

It is worth noting that we consider only a single input graph for the discovery of subgraphs and inference of graph grammar rules. However, discovery of subgraphs and grammar rules across many sample graphs can be done using this approach by viewing the set of sample graphs as one large graph with disconnected subgraphs.

II. Encoding Graphs

Graphs in our setting are directed graphs. More precisely, a graph G is represented by a pair of sets ($\mathcal{V}$, $\mathcal{E}$),

- The set $\mathcal{V}$ of vertices, each has one label selected from a set of $\mathcal{V}_L$ labels.
- The set $\mathcal{E}$ of edges, each is described by its source vertex, destination vertex, and a label drawn from a set of $\mathcal{E}_L$ labels.

A. Proposed Coding Scheme

In this section, we describe a way to encode a directed graph G into a binary string. The proposed coding scheme will be used as a basis for our discussion in the rest of the paper. It is worth noting that the proposed scheme is based on the computation of the minimum description length of a binary representation of a graph provided in [[1], Section 4]. However, we believe that the computation provided in this section is more accurate. As in [1], we describe a directed graph in three steps: $1)$ We first describe the vertices of the graph. In this step, the number of vertices $|\mathcal{V}|$ is first encoded, then their labels are described. $2)$ We then describe the connectivity pattern of the graph, i.e., the source and destination vertices of all edges in the graph. $3)$ Finally, the edge labels are described.

We define $N_1(G)$, $N_2(G)$, and $N_3(G)$ as the number of bits required for the first, second, and third steps, respectively. In the first step, $\log_2^* |\mathcal{V}|$ bits are needed to describe the number of vertices, where the universal coding of positive integers proposed by Rissanen is considered, and for any integer N,

$$\log_2^* N \triangleq \lceil \log_2 N \rceil + \lceil \log_2 \lceil \log_2 N \rceil \rceil + \dots .$$

The label of each vertex can be described using $\lceil \log_2 \mathcal{V}_L \rceil$ bits. The total number of bits required for the first step is,

$$N_1(G) = \log_2^* |\mathcal{V}| + |\mathcal{V}| \lceil \log_2 \mathcal{V}_L \rceil \tag{1}$$

In order to describe the connectivity pattern of a graph, we consider the $|\mathcal{V}|$ x $|\mathcal{V}|$ binary adjacency matrix A, where the entry $A_{i,j}$ in the i^{th} row and the j^{th} column, is 1 if and only if there exists an edge originating at the i^{th} vertex in $\mathcal{V}$ and ending at the j^{th} vertex. We assume that each row of the adjacency matrix has few one entries with respect to the total number of vertices. This assumption can be justified in many applications. For example, in image processing applications, where vertices represent pixels and edges represent correlations between pixels, each pixel is expected to be correlated with only a small number of neighboring pixels that belong to the same object. Based on this assumption, we consider a similar scheme to that described in [[5], Section 4.2] to describe the positions of ones entries in the adjacency matrix.

For each $i \in \{1, 2, \ldots, |\mathcal{V}|\}$, let k_i denote the number of one entries in the i^{th} row of the adjacency matrix, i.e.,

$$k_i = |\{j \in \{1, 2, \ldots, |\mathcal{V}|\} : A_{i,j} = 1\}|. \tag{2}$$

The above assumption can be formally stated as $k_i \ll |\mathcal{V}|, \forall i \in \{1, 2, \ldots, |\mathcal{V}|\}$. We first describe the number k_i for each row in the adjacency matrix, then describe the positions of the one entries within each row. Let $b = \max_i k_i$, then the number of one entries in each row can be described using $\lceil \log_2(b+1) \rceil$ bits. Within the i^{th} row, there are $\binom{|\mathcal{V}|}{k_i}$ possibilities for the positions of k_i one entries, and hence, $\left\lceil \log_2 \binom{|\mathcal{V}|}{k_i} \right\rceil$ bits suffice to describe these positions. We further need to describe the number of bits required to describe the numbers $k_i, i \in \{1, 2, \ldots, |\mathcal{V}|\}$, i.e., we need to encode the integer $\lceil \log_2(b+1) \rceil$. By using universal coding of integers, we know that the above mentioned number is $\log_2^* \lceil \log_2(b+1) \rceil$. The total number of bits required for the second step is,

$$\begin{aligned} N_2(G) &= \log_2^* \lceil \log_2(b+1) \rceil + \sum_{i \in \{1,2,\ldots,|\mathcal{V}|\}} \lceil \log_2(b+1) \rceil + \left\lceil \log_2 \binom{|\mathcal{V}|}{k_i} \right\rceil \\ &= \log_2^* \lceil \log_2(b+1) \rceil + |\mathcal{V}| \lceil \log_2(b+1) \rceil + \sum_{i \in \{1,2,\ldots,|\mathcal{V}|\}} \left\lceil \log_2 \binom{|\mathcal{V}|}{k_i} \right\rceil \end{aligned} \tag{3}$$

Finally, for the third step, $\lceil \log_2 \mathcal{E}_L \rceil$ bits are required to describe the label of each edge. Note that the total number of edges $|\mathcal{E}| = \sum_{i \in \{1,2,\ldots,|\mathcal{V}|\}} k_i$, and hence, the total number of bits required for the third step is,

$$N_3(G) = \lceil \log_2 \mathcal{E}_L \rceil \sum_{i \in \{1,2,\ldots,|\mathcal{V}|\}} k_i \tag{4}$$

The description length of the graph G is defined as,

$$DL(G) \triangleq N_1(G) + N_2(G) + N_3(G) \tag{5}$$

B. Alternative Encodings

In order to justify the choice of the encoding scheme for the graph adjacency matrix in the second step above, we assumed that $k_i \ll |\mathcal{V}|, \forall i \in \{1, 2, \ldots, |\mathcal{V}|\}$. If that assumption does not hold, then a clear alternative would be

to dedicate one bit for each entry in the adjacency matrix, to hold its binary value. Let $N_2^{(2)}(G)$ denote the number of bits required for the second step using the alternative approach, then,

$$N_2^{(2)}(G) = |\mathcal{V}|^2 \tag{6}$$

Now consider the computation of $N_2(G)$ given in (3). The term $\left\lceil \log_2 \binom{|\mathcal{V}|}{k_i} \right\rceil$ is of $O(|\mathcal{V}|)$ for any row i such that the number of non-zero entries (k_i), as well as the number of zero entries ($|\mathcal{V}| - k_i$), are of $O(|\mathcal{V}|)$. If the number of such rows is of $O(|\mathcal{V}|)$, then the suggested alternative can be used.

Consider the other extreme where the following stronger sparsity condition applies on the adjacency matrix,

$$\sum_{i \in \{1,2,\ldots,|\mathcal{V}|\}} k_i \ll |\mathcal{V}|. \tag{7}$$

In other words, the total number of edges of the graph is much less than the number of vertices. For this case $N_2(G)$ is of $O(|\mathcal{V}|)$. Consider a coding scheme that describes the adjacency matrix through a list of (source,destination) pairs for all edges present in the graph. Let $N_2^{(3)}(G)$ denote the number of bits required to describe the adjacency matrix using this scheme, then,

$$N_2^{(3)}(G) = 2\lceil \log_2 |\mathcal{V}| \rceil \sum_{i \in \{1,2,\ldots,|\mathcal{V}|\}} k_i, \tag{8}$$

which is less than $N_2(G)$ under the strong sparsity assumption.

We finally note the in the above coding schemes, we assumed that the graph G is *truly random*. More precisely, there is no prior information about any of the entities being described, that favors one of its possible instances over another.

III. Substructure Discovery

Consider the example graph in Figure 1, and note the repetition of certain patterns (subgraphs). Now, consider a direct application of the coding scheme described in Section II-A. The set of vertices $\mathcal{V}$ has 12 elements, each with a label selected from one of four labels, i.e., $\mathcal{V}_L = 4$, and hence,

$$N_1(G) = \log_2^*(12) + 12\log_2(4) = 30 \tag{9}$$

Note that in the universal coding scheme of positive integers, the first two bits represent one of the integers $\{1, 2, 3, 4\}$, and hence, the representation of twelve is 111011 where the first two bits are the binary representation of three, and hence signify that the following four bits shall be decoded (since 0 is not represented, the binary representation of an integer J is used to denote $J+1$), and the last four bits are the binary representation of eleven, and hence, denote twelve.

For the encoding of the adjacency matrix, we note that each row of the adjacency matrix has only one non-zero entry, except the last row (corresponding to the rightmost vertex) whose entries are all zeros, and hence, $k_i = 1, \forall i \in \{1, 2, \ldots, 11\}$ and $k_{12} = 0$. It follows that $b = 1$, and,

$$N_2(G) = \log_2^*(1) + 12 + 11\lceil \log_2(12) \rceil = 58 \tag{10}$$

Also, since all edges have no labels, we eliminate the third step of the coding scheme, i.e., $N_3(G) = 0$. We conclude that for the graph in Figure 1, the description length - as defined in (5) - is $DL(G) = 88$ bits.

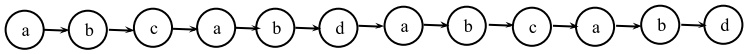


Fig. 1: Graph G with repeated subgraphs

In the following, we provide alternative descriptions of the graph G in Figure 1 in terms of its repeated subgraphs. First consider the subgraph $G1$ in Figure 2a, its set of vertices has two elements, each has a label from a set of two labels, and hence,

$$N_1(G1) = \log_2^*(2) + 2 = 4 \tag{11}$$

As in the description of G above, the number $b = 1$. Moreover $k_1 = 1$ and $k_2 = 0$, as the first vertex 'a' is the source of exactly one edge, and no edges originate at the second vertex. It follows that,

$$N_2(G1) = \log_2^*(1) + 2 + 1 = 5, \tag{12}$$

and hence, $DL(G1) = 9$ bits. Now consider the description of the graph G in terms of its subgraph $G1$, we can easily see that,

$$N_1(G|G1) = \log_2^*(8) + 8\text{x}2 = 21. \tag{13}$$

Also,

$$N_2(G|G1) = \log_2^*(1) + 8 + 7\text{x}3 = 31, \tag{14}$$

and hence, $DL(G|G1) = 52$ bits. In particular, the total description length using the alternative approach equals the sum of the description length of $G1$ and that of G in terms of $G1$. Let $DL^{(2)}(G)$ be the description length of G using the representation in Figure 2, then,

$$DL^{(2)}(G) = DL(G1) + DL(G|G1) = 9 + 52 = 61. \tag{15}$$

We now note that $DL^{(2)}(G) < DL(G)$. The identification of the subgraph $G1$ and its repeated occurrence in G allowed for a more concise description of the graph G that takes only 61 bits instead of the 88 bits required for its original representation.

(a) Graph $G1$ (b) Graph G

Fig. 2: Alternative representation of the graph G in terms of subgraph $G1$

We now show an even more concise description of the graph G, that relies on the identification of the graph $G2$ depicted in Figure 3b. Consider the description of $G2$ in terms of its subgraph $G1$,

$$N_1(G2|G1) = \log_2^*(4) + 4\text{x}2 = 10, \tag{16}$$

$$N_2(G2|G1) = \log_2^*(1) + 4 + 3\text{x}2 = 12, \tag{17}$$

and hence, $DL(G2|G1) = 22$ bits. Now, to describe G in terms of $G2$, we note that the set of labels contains only one element. It follows that all vertices of the graph will have the same label, and there is no need to describe it. We see that,

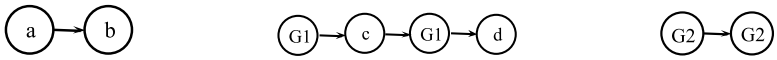
$$N_1(G|G2) = \log_2^*(2) = 2, \tag{18}$$

$$N_2(G|G2) = \log_2^*(1) + 2 + 1 = 5. \tag{19}$$

Let $DL^{(3)}(G)$ denote the description length of the representation the graph G given in Figure 3, then,

$$DL^{(3)}(G) = DL(G1) + DL(G2|G1) + DL(G|G2) = 9 + 22 + 7 = 38 \tag{20}$$

We finally note that $DL^{(3)}(G) < DL^{(2)}(G) < DL(G)$. In other words, the discoveries of subgraphs $G1$ and $G2$ allowed us to compress the representation of the graph G without incurring losses.



(a) Graph $G1$ (b) Graph $G2$ (c) Graph G

Fig. 3: Alternative representation of the graph G in terms of subgraph $G2$

The task of subgraph discovery can be decomposed into the following two steps: 1) Finding candidate subgraphs or subgraph hierarchies, in terms of which, the considered graph will be represented. In the example above, the subgraph hierarchy had graph $G1$ at its root, and then the graph $G2$ was represented in terms of $G1$, and then the graph G was represented in terms of $G2$. 2) Comparing between the candidate subgraph hierarchies, and choosing the one minimizing the description length of the considered graph.

In the SubdueGL algorithm described in [4], an iterative treatment of the above steps is proposed, where in each iteration, a single subgraph is discovered, and replaced by one vertex. In the first step of each iteration, a single vertex is arbitrarily selected, and extended in different ways to form various candidate subgraphs. Only subgraphs that occurs more than once are considered as candidates. Furthermore, the possibilities for extending the selected node are limited by the application of a heuristic. For each candidate subgraph S of a graph G, the following value function is computed,

$$v(S) = DL(S) + DL(G|S). \tag{21}$$

In the second step of the iteration, the subgraph minimizing this value function among all possible candidates (ties are broken arbitrarily), is selected and replaced by a single vertex in the next iteration. The algorithm stops when no candidate subgraphs are found in the first step of the current iteration.

We adopt the same iterative approach, as it reduces the search space for candidate subgraphs. However, we propose an alternative way than that of SubdueGL for finding candidate subgraphs in each iteration. Consider the best representation in Figure 3 for the example graph G considered above, and note that the total number of edges

in all three graphs in Figure 3 is five, the same as the number of edges in the original representation of the graph G in Figure 1 with a unique pair of (source,destination) labels. In general, the number of edges with a unique pair of (source,destination) labels is a lower bound on the total number of edges in any representation of the graph, as each of those edges has to be described in any representation of the graph. Based on this observation, we propose the following algorithm,

Algorithm 1 Subgraph Discovery

1: Consider edges in the graph where one of the following conditions apply:
   - $C1$) The source vertex is a source to all other edges connected to it, and the destination vertex is a destination to all other edges connected to it.
   - $C2$) The source vertex is a destination to all other edges connected to it, and the destination vertex is a source to all other edges connected to it.
2: Each edge in the graph that is considered in the first step, is described by the labels of its source and destination, its edge label, and a one bit flag that indicates whether condition $C1$ or condition $C2$ holds for that edge.
3: The above description of edges may result in repetitions in the description of edges, we keep only one instance of each unique edge description, together with an encoding of the number of its occurrences in the graph.
4: If each edge description in the second step occurs only once, then go to step 9.
5: The representation with the highest number of occurrences is selected (ties are broken arbitrarily).
6: We form a subgraph of two vertices, with labels corresponding to the pair of (source,destination) labels in the selected representation. In the new subgraph, an edge originates at the vertex with the source label and ends at the vertex with the destination label. The new edge will carry the same label as the edge in the representation selected in the previous step. Also, we attach to the description of the new subgraph, the same one bit flag of the edge representation selected in the previous step. Furthermore, we attach to the description of each subgraph a one bit *mark* that is either *active* or *inactive*. We activate the mark for the new subgraph. If any of the vertices constituting the new subgraph is a subgraph itself, then we deactivate the mark attached to the description of the corresponding subgraph.
7: A unique new label is assigned to the new subgraph. Each instance of the new subgraph in the original graph is replaced with a single vertex that is given the new label.
8: Return to step 1.
9: If there is no saved subgraphs with an active mark, then the algorithm stops. Otherwise, one subgraph with an active mark is selected, its mark is deactivated, and we set this selected subgraph as the graph considered by the algorithm, then return to step 1.

Note that the role of the one bit flag attached to each discovered subgraph, is to indicate which of the two vertices of the subgraph is destination (source) to edges ending (originating) at the new vertex replacing the subgraph in the compressed representation of the original graph. In the Appendix, we illustrate how the algorithm works for

the example graph G, given in Figure 1, to reach the representation in Figure 3.

We also note the role of the one bit mark attached to each subgraph, in eliminating separate representations of trivial subgraphs that occur only once in another subgraph. This role is highlighted in the trace provided in the Appendix through the transition from the representation in Figure 6 to that in Figure 3. However, once the algorithm stops, the one bit marks can be removed since they are not needed to reconstruct the graph. It can be verified that since all saved subgraphs are nontrivial, each of them reduces the total description length of the input graph.

IV. Learning Graph Grammar: A Probabilistic Approach

The discovery of repeated substructures is not only useful for finding a more concise description of the original structure. Discovered patterns can be meaningful for the considered application. For example, in image processing, a repeated pattern of colors of a set of pixels having the same orientation, may indicate the occurrence of more than one instance of a certain object. Furthermore, those instances may occur with slight differences, e.g. because of independent background factors that are considered as noise. In this section, we discuss the extraction of patterns from a sample graph. Unlike the subgraph discovery algorithm presented in the previous section, the task here is to identify patterns that apply to other graphs that are similar but not necessarily identical to the considered sample. We say that those general patterns define grammar rules for a language that describes a type of graphs, to which the sample graph belongs.

Consider the graphs $G4a$ and $G4b$, depicted in Figure 4, and their similarity to the graph G in Figure 1. In particular, Each of the three graphs consists of connected chains, where each chain has either the form $a \rightarrow b \rightarrow c$ or the form $a \rightarrow b \rightarrow d$. In order for a graph grammar rule to formally express this general structure, we need to introduce the following,

- **Variables:** Vertices can have variable labels that can take one of many specified values. In the considered example, each chain has the form $a \rightarrow b \rightarrow v$, where v is a variable node with a label selected from the set $\{c, d\}$. We represent this rule graphically in Figure 5a, where the sign $|$ means *Or*.
- **Recursion Rules:** Different instances of subgraphs satisfying the same grammar rule can have a different number of repetitions of subgraphs satisfying another grammar rule. In the considered example, the graphs in Figures 1, 4a, and 4b, have four, three, and three repetitions of subgraphs satisfying the rule $R1$ depicted in Figure 5a, respectively. We represent this rule graphically in Figure 5b, where the black filled circle denotes a stop sign, and for any grammar rule Rx, G_{Rx} denotes any graph satisfying the rule Rx. Any graph satisfying the rule $R2$ consists of a chain of an arbitrary number of subgraphs, each satisfying the rule $R1$.

(a) Graph $G4a$ (b) Graph $G4b$

Fig. 4: Graphs similar to the graph G of Figure 1

(a) Rule $R1$ (b) Rule $R2$

Fig. 5: Grammar rules deduced by the SubdueGL algorithm [4] for the graphs in Figures 1, 4a, and 4b

In [4], the SubdueGL algorithm considers both above aspects, and deduces the grammar rule $R2$ for the example graph G of Figure 1. We do not discuss the details of the implementation of SubdueGL here. However, we note that in SubdueGL, the two aspects above (Variables and Recursion Rules) are the only additions to those considered by a subgraph discovery algorithm (e.g. Algorithm 1). In other words, the generalization from discovering subgraphs akin to the ones in Figure 3 to deducing grammar rules like the ones in Figure 5 is done only through the consideration of those two new aspects. Based on this note, we now discuss a shortcoming of SubdueGL and propose a framework that deals with it.

Although the graphs G, $G4a$, and $G4b$, depicted in Figures 1, 4a, and 4b, respectively, are similar, each of them suggests a different frequency for different instances of the variable node in grammar rule $R1$. Also, each of those example graphs suggests a different likelihood for the occurrence of the repetition in the rule $R2$. More precisely, each of those examples has a different empirical probability for the elements inside the dashed squares in Figure 1.

In making the transition from discovering subgraphs that exactly describe an example graph, akin to the ones in Figure 3, to deducing general grammar rules that generalize to similar but different graphs, as the ones in Figure 5, we lose information about the exact description of the graph, and keep only those information that describe its structure. However, the definition of the structure of the graph is subject to the goal of the overlying application. For example, An application may consider a graph consisting of a chain of 100 repetitions of subgraphs satisfying $R1$, among which 99 have the variable node labeled 'c' and one has it labeled 'd', **dissimilar** from another graph consisting of a chain of 100 repetitions of subgraphs satisfying $R1$, 50 of them have the variable node labeled 'c' and 50 have it labeled 'd'. Furthermore, both graphs can represent different objects from another that consists only of a chain of two repetitions of subgraphs satisfying $R1$. In that case, SubdueGL will erroneously deduce the same rule $R2$ of Figure 5b for all three graphs.

We propose a probabilistic framework that assigns a probability to each possible outcome of an *Or* sign in the rules deduced by SubdueGL. Moreover, in order to allow for context-based inference of grammar rules where prior statistical information is available about the language, we assign a prior probability to each possible grammar rule. Let $\mathcal{R}$ denote the alphabet of all possible grammar rules. We categorize grammar rules in $\mathcal{R}$ into classes, and assign a prior probability to each class. Given a graph example Gx, the deduced grammar rule is,

$$R = \text{argmax}_{R\in\mathcal{R}} \quad p(R, Gx) = p(C_R)p(Gx|R), \tag{22}$$

where C_R denotes the class containing the rule R, and the conditional probability $p(Gx|R)$ equals the product of the probabilities assigned in R to instances of variable vertices in Gx, as well as probabilities assigned to instances

of recursion rules. An equal probability of priors over all classes of rules in the alphabet $\mathcal{R}$ will result in a selection of the grammar rule that *most fits* the input example. For example, if there is a class of grammar rules containing only those rules that have neither variables nor recursion rules, and equal priors of all classes are assigned, then given an input example, it is always the case that a grammar rule belonging to the aforementioned class is selected, a rule for which the conditional probability $p(Gx|R)$ is one. In this case, we lose the generalization advantage and the selected rule is *custom tailored* to the input example, i.e., the conditional probability $p(Gy|R)$ for any different example graph Gy is zero. For this reason, it is desired to have an unequal prior probability of rule classes, one that favors more general rules, e.g. rules with more variables and recursion rules. This issue can be viewed as a bias-variance tradeoff where the role of priors is to minimize the variations between possible instances of the deduced grammar rule, and the role of the conditional probability is to minimize the bias of the deduced grammar rule towards graphs that are dissimilar from the input graph. Finally, it is worth noting that the maximum entropy principle discussed in [6] can be used to set the a priori probabilities of grammar rule classes based on available knowledge.

V. Conclusion and Future Work

Given an input graph, we considered the problem of discovering repeated subgraphs, and finding a concise representation of the graph in terms of the discovered subgraphs. For this purpose, we proposed a graph binary encoding scheme, that is similar to that introduced in [1]. The provided computation of the description length of graphs using the proposed coding scheme is more accurate than that in [1]. We then proposed a subgraph discovery iterative algorithm, that discovers a new subgraph in each iteration. The proposed algorithm then removes trivial subgraphs that occur only once in another discovered subgraph. Although there are no optimality guarantees provided for the proposed algorithm, it can be easily seen that after the algorithm stops, each saved subgraph reduces the total description length of the graph. Furthermore, we provide a simple example for which the proposed algorithm finds the most concise representation of the input graph with respect to the proposed graph encoding scheme.

In Section IV, we discussed the generalization of subgraph discovery to inferring graph grammar rules that apply to a class of graphs containing the input graph as well as other similar but different graphs. We reviewed the aspects considered by the SubdueGL algorithm in [4]. Furthermore, we introduced a general probabilistic framework that is expected to be useful in applications where SubdueGL deduces the same grammar rules for input graphs that are considered by the application as dissimilar. A clear advantage of the probabilistic approach is the clear control over the bias-variance tradeoff through the setting of a priori probabilities of grammar rules as well as the conditional probabilities of instance graphs given a grammar rule.

In future work, we plan to pursue a subgraph discovery algorithm that guarantees a minimal representation of the input graph with respect to the coding scheme in Section II. Also, it is not clear to us why the possibility of variable edges was not considered in SubdueGL, and only variable vertices were considered. We plan to gain a deeper understanding of the impact of the introduction of variable edges to the inference of grammar rules. Finally, as in SubdueGL, we plan to investigate the possibility of embedding context-based rules in the probabilistic framework.

As an example, the connectivity of instance subgraphs satisfying a grammar rule can control the prior probability of that rule. Formal measures for connectivity of graphs that can be useful for this purpose were introduced through the notions of graph entropy in [7] and [8], and the graph guessing number discussed in [9] and [8].

REFERENCES

[1] D. Cook, and L. Holder "Substructure Discovery Using Minimum Description Length and Background Knowledge," *Journal of Artificial Intelligence Research,* vol. 1, Aug. 1993.

[2] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, pp. 465-471, 1978.

[3] P. Niyogi, "The Informational Complexity of Learning: Perspectives on Neural Networks and Generative Grammar," *First Edition, Springer*, Nov. 1997.

[4] I. Jonyer, L. Holder and D. Cook, "Concept formation using graph grammars," *In Proc. KDD Workshop on Multi-Relational Data Mining*, 2002.

[5] J. R. Quinlan, L. R. Rivest "Inferring decision trees using the minimum description length principle," *Information and Computation* 80, pp.227-248, 1989.

[6] A. Berger, V. Pietra, and S. Pietra "A maximum entropy approach to natural language processing," *Computational linguistics*, vol. 22, no. 1, pp. 39-71, 1996.

[7] A. Rao "Entropy of a graph," *Lecture Notes: Information Theory in Computer Science, available at: http://www.cs.washington.edu/homes/anuprao/pubs/CSE533Autumn2010/lecture4.pdf*, Oct. 2010.

[8] S. Riis "Graph entropy, network coding, and guessing games," *ArXiv preprint, available at: http://arxiv.org/abs/0711.4175* Nov. 2007.

[9] S. Riis "Information flows, graphs and their guessing numbers ," *In Proc.* 4^{th} *International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks* pp.1-9, Apr. 2006.

[10] J. Coble, R. Rathi, D. Cook, and L. Holder "Iterative structure discovery in graph-based data," *International Journal of Artificial Intelligence Techniques*, 2005.

[11] L. Holder, D. Cook, and H. Bunke "Fuzzy Substructure Discovery," *In Proc.* 9^{th} *International Conference on Machine Learning,* pp. 218-223, 1992.

[12] A. Barron, J. Rissanen, and B. Yu, "The minimum description length principle in coding and modeling," *IEEE Trans. Information Theory,* vol. 44, no. 6, pp. 2743-2760, 1998.

APPENDIX

TRACE OF ALGORITHM 1 WITH THE INPUT GRAPH G OF FIGURE 1

We first provide a detailed step by step trace of the first iteration of the algorithm.

- In step 1, all edges in the graph in Figure 1 are considered, since the graph is a chain and all edges satisfy the condition $C2$.
- Steps 2 and 3 result in the following edge descriptions: $E1$: An edge from a vertex with label 'a' to a vertex with label 'b', occurring four times, $E2$: An edge from a vertex with label 'b' to a vertex with label 'c', occurring two times, $E3$: An edge from a vertex with label 'c' to a vertex with label 'a', occurring two times, $E4$: An edge from a vertex with label 'b' to a vertex with label 'd', occurring two times, $E5$: An edge from a vertex with label 'd' to a vertex with label 'a', occurring one time. All edges have no labels and their attached one bit flags indicate that they satisfy condition $C2$.
- The algorithm proceeds to step 5 since descriptions $E1$, $E2$, $E3$, and $E4$, occur more than once.
- The representation $E1$ has the highest number of occurrences, and hence, is selected at step 5.

- In step 6, the new subgraph is the subgraph depicted in Figure 2a and the attached one bit flag points to condition $C2$. The mark attached to the new subgraph is activated.
- In step 7, the compressed representation of the original graph is the one depicted in Figure 2b.

We now trace the results of the following iterations.

- After four iterations of the algorithm, we obtain the subgraphs in Figure 6. The marks attached to the descriptions of subgraphs $G1$, $G2$, and $G3$, are deactivated, since each of them occur in another subgraph. Subgraph $G4$ is the only subgraph with an active mark.
- In the fifth iteration, the algorithm jumps from step 4 to step 9, where subgraph $G4$ is set as the considered graph and the algorithm returns to step 1.
- In the sixth iteration, the subgraph in Figure 2a is rediscovered as a subgraph of $G4$. We then obtain the representation in Figure 3. Note that now, the subgraph in Figure 3a is the only subgraph carrying an active mark.
- In the seventh iteration the algorithm jumps from step 4 to step 9 since the edges in the graph depicted in Figure 3b are all *unique*. In step 9, the graph in Figure 3a is set as the considered graph and the algorithm returns to step 1.
- In the eighth iteration, the algorithm jumps from step 4 to step 9, since all the edges in the considered graph in Figure 3a are all unique. In step 9, the algorithm stops since the marks attached to all saved subgraphs are now inactive.

(a) Graph $G1$ (b) Graph $G2$ (c) Graph $G3$ (d) Graph $G4$ (e) Graph G

Fig. 6: Temporary representation of the graph G After four iterations of Algorithm 1