Inference of Node and Edge Replacement Graph Grammars

Jacek P. Kukluk
Dept. of Computer Science and Engineering
University of Texas at Arlington
Jkukluk@gmail.com

Lawrence B. Holder, Diane J. Cook
School of Electrical Engineering and Computer Science
Washington State University
holder@wsu.edu, cook@eecs.wsu.edu

Abstract

In this paper we study the inference of node and edge replacement graph grammars. We search for frequent subgraphs and then check for overlap among the instances of the subgraphs in the input graph. If subgraphs overlap by one node, we propose a node replacement graph grammar production. If subgraphs overlap by two nodes or two nodes and an edge, we propose an edge replacement graph grammar production. We also can infer a hierarchy of productions by compressing portions of a graph described by a production and then inferring new productions on the compressed graph. We validate the approach in experiments where we generate graphs from known grammars and measure how well the approach infers the original grammar from the generated graph. We show the graph grammars found in biological molecules, biological networks and analyze learning curves of the algorithm.

1. Introduction

Noam Chomsky (1956) pointed out that one of the main concerns of a linguist is to discover simple grammars for natural languages and study those grammars with the hope of finding a general theory of linguistic structure. While string grammars represent language, we are looking for graph grammars that represent graph properties and can generalize these properties from finite graph examples into generators that can generate an infinite number of graphs. String grammars can be inferred from a finite number of sentences and generalize to an infinite number of sentences. Inferring graph grammars will generalize the knowledge from the examples into a concise form and generalize to an infinite number of entities from the domain.

We study the inference of node and edge replacement graph grammars. We search for frequent subgraphs and then check for overlap among the instances of the subgraphs in the input graph. If subgraphs overlap by one node, we propose a node replacement graph grammar production. If subgraphs overlap by two nodes or two nodes and an edge, we propose an edge replacement graph grammar production. We also can infer a hierarchy of productions by compressing portions of a graph described by a production and then inferring new productions on the compressed graph. We validate the approach in experiments where we generate graphs from known grammars and measure how well the approach infers the original grammar from the generated graph. We show the graph grammars found in biological molecules and biological networks and analyze learning curves of the algorithm

2. Related work

A vast amount of research has been done in inferring grammars. These analyses focus on string grammars where symbols appear in a sequence. We are concerned with graph grammars, which can represent much larger classes of problems than string grammars. Only a few studies can be found in graph grammar inference.

Jeltsch and Kreowski (1990) did a theoretical study of inferring hyperedge replacement graph grammars from simple undirected, unlabeled graphs. Their paper leads through an example where from four complete bipartite graphs (K3,1;,K3,2; K3,3; K3,4), the authors describe the inference of a grammar that can generate a more general class of bipartite graphs (K3,n), where $n \geq 1$. The authors define four operations that lead to a final hyperedge replacement grammar. Jeltsch and Kreowski start the process from a grammar which has all the sample graphs in its productions. Then they transform the initial productions into productions that are more general but can still produce every graph from the sample graphs. Their approach guarantees that the final grammar will generate graphs that contain all sample graphs.

Oates, Doshi, and Huang (2003) discuss the problem of inferring probabilities of every grammar rule for stochastic hyperedge replacement context free graph grammars. They call their program Parameter Estimation for Graph Grammars (PEGG). They assume that the grammar is given. Given a structure of a grammar S and a

finite set of graphs E generated by grammar S, they ask what are the probabilities θ associated with every rule of the grammar. Their strategy is to look for a set of parameters θ that maximizes the probability p(E| S, θ).

In terms of similarity to string grammar inference we consider the Sequitur system developed by Nevill-Manning and Witten (1997). Sequitur infers a hierarchical structure by replacing substrings based on grammar rules. The new, compressed string is searched for substrings which can be described by the grammar rules, and they are then compressed with the grammar and the process continues iteratively. Similarly, in our approach we replace the part of a graph described by the inferred graph grammar with a single node and we look for grammar rules on the compressed graph and repeat this process iteratively until the graph is fully compressed.

Jonyer et al.'s approach to node-replacement graph grammar inference (Jonyer, Holder, and Cook, 2002, 2004) starts by finding frequently occurring subgraphs in the input graphs. They check if isomorphic instances of the subgraphs that minimize the measure are connected by one edge. If they are, a production S→ PS is proposed, where P is the frequent subgraph. P and S are connected by one edge. Jonyer's method of testing if subgraphs are adjacent by one edge limits his grammars to descriptions of "chains" of isomorphic subgraphs connected by one edge. Since an edge of a frequent subgraph connecting it to the other isomorphic subgraph can be included to the subgraph structure, testing subgraphs for overlap allows us to propose a class of grammars that have more expressive power than the graph structures covered by Jonyer's grammars. For example, testing for overlap allows us to propose grammars which can describe tree structures, while Jonyer's approach does not allow for tree grammars.

3. Definitions

We give the definition of a graph and a graph grammar which is relevant to our approach and the implemented system. The defined graph has labels on vertices and edges. Every edge of the graph can be directed or undirected. The definition of a graph grammar describes the class of grammars that can be inferred by our approach. We emphasize the role of recursive productions in the name of the grammar, because the type of inferred productions are such that the non-terminal label on the left side of the production appears one or more times in the node labels of a graph on the right side. This is the main characteristic of our grammar productions. Our approach can also infer non-recursive productions. The embedding mechanism of the grammar consists of connection instructions. Every connection instruction is a pair of vertices that indicate where the production graph can connect to itself in a recursive fashion.

A *labeled graph* G is a 6-tuple, $G=(V,E,\mu,\nu,\eta,L)$, where
V - is the set of nodes,
$E\subseteq V\times V$ - is the set of edges,
$\mu: V\to L$ - is a function assigning labels to the nodes,
$\nu: E\to L$ - is a function assigning labels to the edges,
$\eta: E\to\{0,1\}$ - is a function assigning direction property to edges (0 if undirected, 1 if directed).
L - is a set of labels on nodes and edges.

A *node replacement recursive graph grammar* is a tuple $Gr=(\Sigma,\Delta,\Gamma,P)$, where
Σ - is an alphabet of node labels,
Δ - is an alphabet of terminal node labels, $\Delta\subseteq\Sigma$,
Γ - is an alphabet of edge labels, which are all terminals,
P - is a finite set of productions of the form (d,G,C), where $d\in\Sigma-\Delta$, G is a graph, C is an embedding mechanism with a set of connection instructions, $C\subseteq V\times V$, where V is the set of nodes of G. A connection instruction $(v_i,v_j)\in C$ implies that derivation can take place by replacing v_i in one instance of G with v_j in another instance of G. All the edges incident to v_i are incident to v_j. All the edges incident to v_j remain unchanged.

An *edge replacement recursive graph grammar* is a 5-tuple $Gr=(\Sigma,\Delta,\Gamma,\Omega,P)$, where
Σ - is an alphabet of node labels,
Δ - is an alphabet of terminal node labels, $\Delta\subseteq\Sigma$,
Γ - is an alphabet of edge labels,
Ω -is an alphabet of terminal edge labels, $\Omega\subseteq\Sigma$,
P - is a finite set of productions of the form (d,G,C), G is a graph, where $d\in\Gamma-\Omega$, C is an embedding mechanism with a set of connection instructions, $C\subseteq(V\times V;V\times V)$, where V is the set of nodes of G. A connection instruction $(v_i,v_j;v_k,v_l)\in C$ implies that derivation can take place by replacing v_i, v_k in one instance of G with v_j,v_l respectively, in another instance of G. All the edges incident to v_i are incident to v_j, and all the edges incident to v_k are incident to v_l. All the edges incident to v_j and v_k remain unchanged. If, in derivation process after applying connection instruction $(v_i,v_j;v_k,v_l)$, nodes v_i,v_j are adjacent by an edge, we call edge $e=(v_i,v_j)$ a *real edge*, otherwise edge $e=(v_i,v_j)$ is used only in the specification of the grammar and we call this edge a *virtual edge.* We introduce the definition of two data structures used in our algorithm.

A *substructure S* of a graph *G* is a data structure which consists of: (1) graph definition of a substructure S_G which is a graph isomorphic to a subgraph of *G*, (2) list of instances (I_1, I_2, …, I_n) where every instance is a subgraph of *G* isomorphic to S_G.

A *recursive substructure recursiveSub* is a data structure which consists of:

(1) graph definition of a substructure S_G which is a graph isomorphic to a subgraph of *G*

(2) list of connection instructions which are pairs of integer numbers describing how instances of the substructure can overlap to comprise one instance of the corresponding grammar production rule.
(3) List of recursive instances $(IR_1, IR_2, \ldots, IR_n)$ where every instance IR_k is a subgraph of G. Every instance IR_k consist of one or more isomorphic copies of S_G, overlapping by no more than one vertex in the algorithm for node graph grammar inference and no more than two vertices in edge grammar inference.

.

In our definition of a substructure we refer to subgraph isomorphism. However, in our algorithm we are not solving the subgraph isomorphism problem. We are using a polynomial time beam search to discover substructures and graph isomorphism to collect instances of the substructures.

4. The Graph Grammar Inference Algorithms

An example in Figure 1 shows a graph composed of three overlapping substructures. The algorithm generates candidate substructures and evaluates them using any one of the learning biases, which are discussed later. The input to our algorithm is a labeled graph G which can be one connected graph or set of graphs. G can have directed or undirected edges. The algorithm begins by creating a list of substructures where every substructure is a single node and its instances are all nodes in the graph with the same node label. Initially, the best substructure is the node with the most instances. The substructures are ranked and placed on the expansion queue Q. It then extends all substructures in Q in all possible ways by a single edge and a node or only by single edge if both nodes are already in the graph definition of the substructure. We keep all extended substructures in newQ. We evaluate substructures in newQ according to the chosen evaluation heuristic.

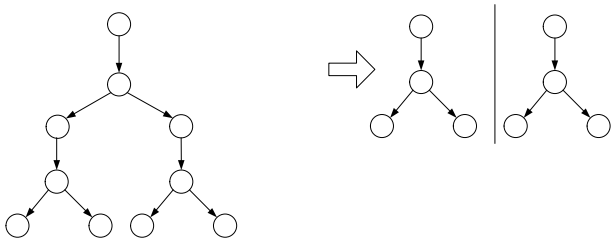

Figure 1: A graph with overlapping substructures and a graph grammar representation of it.

The total number of substructures considered is determined by the input parameter Limit. The best substructure identified becomes the right side of the first grammar production, and the graph G is compressed using this best substructure. Compression replaces every instance of best substructure with a single non-terminal node. This node is labeled with a non-terminal label. The compressed graph is further processed until it cannot be compressed any more, or some user-defined stopping condition is reached (maximum number of productions, for instance). In consecutive iterations the best substructure can have one or more non-terminal labels. It allows us to create a hierarchy of grammar productions. The input parameter Beam specifies the width of the beam search, that is, the length of Q. The Algorithm 1 shows the pseudocode.

Algorithm 1 Graph grammar discovery.

INFER_GRAMMAR (graph *G*, integer *Beam*, integer *Limit*)

1. *grammar*={}
2. **repeat**
3. queue Q ={v | v is a node in G having a unique label}
4. *bestSub*= first substructure in Q
5. **repeat**
6. *newQ* ={}
7. **for each** substructure $S \in Q$
8. *newSubs* = extend substructure S in all possible ways by a single edge and a node
9. *recursiveSub* = RECURSIFY_SUBSTRUCTURE(S)
10. *newQ* = *newQ* $\cup$ *newSubs* $\cup$ *recursiveSub*
11. *Limit*=*Limit*-1
12. evaluate substructures in *newQ* , maintain length(*newQ*) < *Beam* eliminating substructure with the lowest value if necessary
13. **end for**
14. **if** best substructure in *newQ* better than *bestSub*
15. **then** *bestSub* = best substructure in *newQ*
16. Q=*newQ*
17. **until** Q is empty or *Limit* ≤ 0
18. *grammar* = *grammar* $\cup$ *bestSub*
19. G = G compressed by *bestSub*
20. **until** *bestSub* cannot compress the graph G
21. **return** *grammar*

Recursive productions are identified during the previously described search process by allowing instances to grow and overlap. Any two instances are allowed to overlap by only one vertex. The recursive substructure is evaluated along with non-recursive substructures and is competing with non-recursive substructures for placement on Q. Connection instructions are created by determining which nodes overlapped across instances. Figure 2 shows an example of a substructure that is the right side of a recursive rule, along with its connection instructions (Kukluk, Holder, and Cook, 2006).

The edge replacement algorithm operates on a data structure called a substructure (similar to the algorithm for

node replacement grammar inference). A substructure consists of a graph definition of the repetitive subgraph and its instances. We illustrate it in Figure 3. We grow substructures similarly as in the algorithm for node replacement graph grammar inference, then we examine instances for overlap. If two nodes v_1, v_2 in G both belong to two different instances (two overlapping instances), we propose a recursive grammar rule. We determine the type of non-terminal edge. If v_1, v_2 are adjacent by an edge, it is a real edge, and we determine its label which we use to specify the terminating production. If v_1, v_2 are not adjacent, then the non-terminal edge is virtual. In Figure 3 we illustrate how we determine connection instructions.

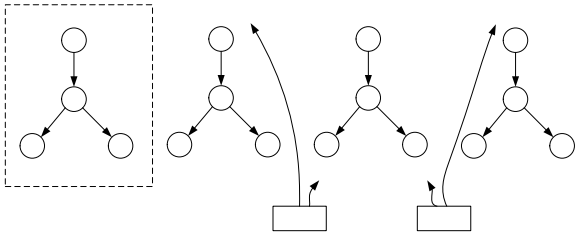

Figure 2: Substructure and its instances while determining connection instructions (continuation of the example from Figure 1).

One advantage of our algorithm is its modular design in which the evaluation of candidate grammar rules is done separately from the generation of these candidates. The result is that any evaluation metric can be used to drive the search. Different evaluation metrics are part of the system and can be specified as arguments. We have had great success with the minimum description length (MDL) principle on a wide range of domains. MDL is an information theoretic approach (Rissanen, 1989). The description length of the substructure S given the input graph G is calculated as DL(S,G) =DL(S)+DL(G|S), where DL(S) is the description length of the subgraph, and DL(G|S) is the description length of the input graph compressed by the subgraph (Cook and Holder, 1994, 2000). An alternative measure is the size heuristic which is computed as

$$\frac{size(G)}{size(S)+size(G \mid S)}$$

where G is the input graph, S is a substructure and G|S is the graph derived from G by compressing each instance of S into a single node. size(t) can be computed simply by summing the number of nodes and edges: size(t) = vertices(t) + edges(t). The third measure is called setcover, which is used for concept learning tasks using sets of disconnected graphs. This measure maximizes the number of positive examples in which the grammar production is found while minimizing the number of such negative examples.

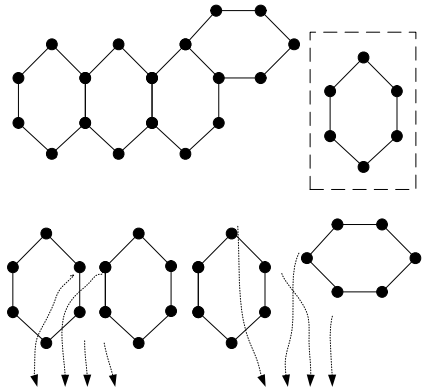

Figure 3. The input graph (a), substructure graph definition (b) and four overlapping instances of repetitive subgraph (c).

Our algorithms make use of the substructure discovery algorithm described in Cook and Holder (2000). This algorithm uses a heuristic search whose complexity is polynomial in the size of the input graph. The overlap test is the main computationally expensive addition of our grammar discovery algorithm and it does not change its complexity. The number of nodes of an instance graph is not larger than V, where V is the number of nodes in the input graph. Checking two instances for overlap will not take more than $O(V^2)$ time. The number of pairs of instances is no more than V^2, so the entire overlap test will not take more than $O(V^4)$ time.

5. Experiments

5.1 Methodology

Having our algorithm implemented, we faced the challenge of evaluating its performance. There are an infinite number of grammars as well as graphs generated from these grammars. We seek to understand the relationship between graph grammar inference and grammar complexity, and so need a measure of grammar complexity. One such measure is the Minimum Description Length (MDL) of a graph, which is the minimum number of bits necessary to completely describe the graph.

In our experiments we measure an error based on structural difference. Another approach to measuring the accuracy of the inferred grammar would be based on a graph grammar parser. We would consider accurate the inferred grammars that can parse the input graph. Graph grammar parser would require subgraph isomorphism test which is computationally expensive and much more difficult in implementation than the error measure we are using. For these reasons we did not pursue implementation of graph grammar parser.

We would like our error to be a value between 0 and 1; therefore, we normalize the error by having in the denominator the sum of the size of the graph used in the original grammar and the number of non-terminals. We do not allow an error to be larger than 1; therefore, we take the minimum of 1 and our measure as a final value. The restriction that the error is not larger than 1 prohibits unnecessary influence on the average error taken from several values by inferred graph structure significantly larger than the graph used in the original grammar.

$$Error = \min\left(1, \frac{\text{matchCost}(g_1, g_2) + |\#CI - \#NT|}{\text{size}(g_1) + \#NT}\right),$$

where

$\text{matchCost}(g_1, g_2)$ is the minimal number of operations required to transform g_1 to a graph isomorphic to g_2, or g_2 to a graph isomorphic to g_1. The operations are: insertion of an edge or node, deletion of a node or an edge, or substitution of a node or edge label.

$\#CI$ is the number of inferred connection instructions

$\#NT$ is the number of non-terminals in the original grammar

$\text{size}(g_1)$ is the sum of the number of nodes and edges in the graph used in the grammar production

5.2 Error as a function of noise and complexity of a graph grammar

We used twenty nine graphs from Figure 5 in grammar productions. We assigned different labels to nodes and edges of these graphs except three nodes used for non-terminals. As noise we added nodes and edges to the generated graph structure. We compute the number of added nodes from the formula (noise/(1-noise))*number_of_nodes. Similary for edges. We generated graphs with noise from 0 to 0.9 in 0.1 increments. For every value of noise and MDL we generated thirty graphs from the known grammar and inferred the grammar from the generated graph. We computed the inference error and averaged it over thirty examples. We generated 8700 graphs to plot each of the three graphs in Figure 4. The first plot shows results for grammars with one non-terminal. The second and the third plot show results for grammars with two and three non-terminals.

We average the value of an error over ten values of noise which gives us the value we can associate with the graph structure. It allowed us to order graph structures used in the grammar productions based on average inference error. In Figure 5 we show all twenty nine connected simple graphs with three, four and five nodes used in productions ordered in non-decreasing MDL value of a graph structure.

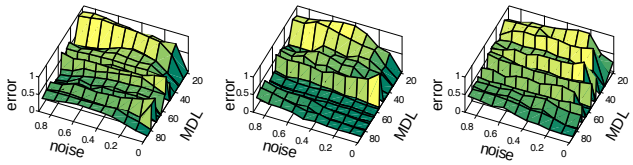

Figure 4: Error as a function of noise and MDL where graph structure was not corrupted (one, two and three non-terminals respectively).

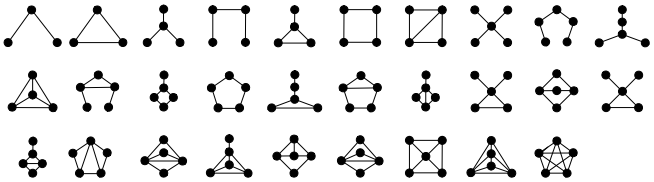

Figure 5: Twenty nine simple connected graphs ordered according to non-decreasing MDL value.

5.3 Error as a function of number of labels

We would like to evaluate how error depends on the number of different labels used in a grammar. We restricted graph structures used in productions to graphs with five nodes. Every graph structure we labeled with 1, 2, 3, 4, 5 or 6 different labels. For every value of MDL and number of labels we generated 30 different graphs from the grammar and computed average error between them and the learned grammars. The generated graphs were without noise. We show the results for one, two, and three non-terminals in Figure 6. Below the three dimensional plots, for clarity, we give two dimensional plots with triangles representing the errors. The larger and lighter the triangle the larger is the error. We see that the error increases as the number of different labels decreases. We see on the two dimensional plots the shift in error towards graphs with higher MDL when the number of non-terminals increases.

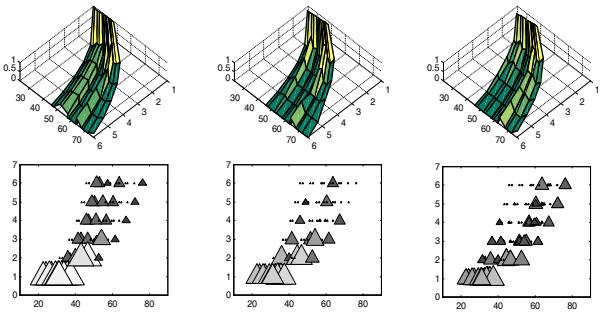

Figure 6 : Error as a function of MDL and number of different labels used in a grammar definition (one, two and three non-terminals respectively).

5.4 Learning Curves

We wanted to examine the learning process on a graph grammar with several productions. Since there are an infinite number of different graph grammars, we decided to select one example with several different graph structures used in the grammar productions. We show this example in Figure 7, where we see the graph grammar used to generate graphs. There are five productions. The last production with only one node is a terminating production. Each graph in the first four productions had two non-terminal nodes. The first four productions are chosen with probability 0.1 in the generation process. The terminating production is chosen with probability 0.6.

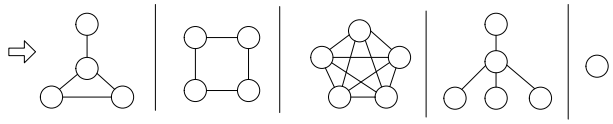

Figure 7: Graph grammar used for graph generation

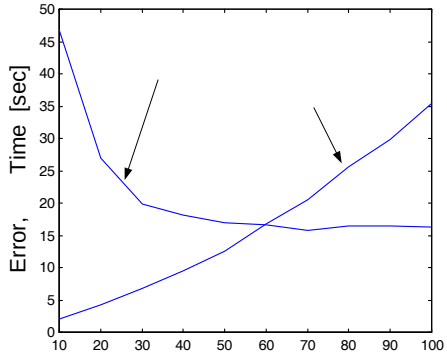

Figure 8: Error and time as a function of number of graphs in the training set.

We generate sets of graphs with 10, 20, 30, and up to 100 graphs generated from the grammar in Figure 7. Every graph in the set has 30 to 40 nodes. We compare the first four grammar productions found by our algorithm to the original grammar in Figure 7. As a measure of an error, we use the minimal match cost of a transformation from one graph structure to the other, as described in section 5.1 where we talk about the measure of the error. We calculate the match cost of the structure of the graph from the first inferred grammar production to the four structures of the original productions and choose the smallest value. Then, we calculate the match cost of the structure from the second inferred production to the three structures from the original grammar not selected before and select the smallest value. Similarly, we find the smallest match cost between the structure of the third inferred production and the two structures left. The last inferred production we compare to the remaining production from the original grammar. The inference error we compute as a sum of the four errors we just explained. We repeat generation and error determination thirty times and compute the average value of the error. In Figure 8 we show the grammar inference error and time as a function of the number of graphs in the input set. We see that time in the range 10 to 100 graphs has close to linear increase. The error decreases sharply as we increase the set of graphs from 10 to 30. The error does not reach zero. The input graph has now four patterns. We often infer productions which contain two of the patterns or a portion of two patterns which causes the error.

5.5 Biological networks

The biological networks used in our experiments were from the Kyoto Encyclopedia of Genes and Genomes. (KEGG) (Kanehisa, et al., 2006). We use a graph representation which has labels on vertices and edges. The graphs represent processes like metabolism, membrane transport, and biosynthesis. We group the graphs into sets which allow us to search for common recursive patterns which can help to understand basic building blocks and hierarchical organization of processes. The label entry represents a molecule, a molecule group or a pathway. A node labeled entry can be connected to a node labeled type. The type can be a value of the set: enzyme, ortholog, gene, group, compound, or map. A reaction is a process where a material is changed to another material catalyzed by an enzyme. A reaction, for example, can have one or more enzyme entries, and one or more compounds. Labels on edges show relationships between entities. The meanings are: Rct_to_P : reaction to Product , S_to_Rct : substrate to reaction, E_to_Rct : enzyme (gene) to reaction, E_to_Rel: enzyme to relation, Rel_to_E: relation to enzyme. Nodes labeled ECrel indicate an enzyme-enzyme relation meaning that two enzymes catalyze successive reactions.

We use ten species in our experiments. The abbreviated names of the species and their meanings are: bsu - Bacillus subtilis, sty - Salmonella enterica serovar Typhi CT18, xcc - Xanthomonas campestris pv. campestris ATCC 33913, pto - Picrophilus torridus, mka - Methanopyrus kandleri, pho - Pyrococcus horikoshii, sfx - Shigella flexneri 2457T (serotype 2a), efa - Enterococcus faecalis, bar - Bacillus anthracis Ames 0581,

The species we selected randomly from the database. The number of networks is different for each species. We wanted to see how our algorithm performs when we increase sample size of graphs supplied to our inference algorithm. For this purpose we divided all the networks into 11 sets such that the last set (11th) has all the species. Set 10 excludes the 11th portion of all networks. Set 9 excludes 2/11 of all networks and set 1 has 1/11 of all networks. If all networks in the species do not divide by 11 evenly we distribute the remaining networks randomly to the 11 sets.

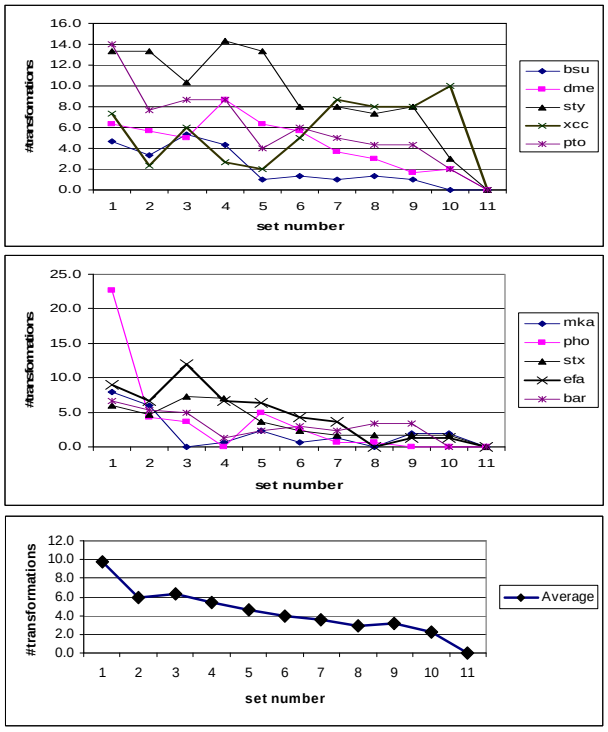

Figure 9: Change in inferred grammar measured in reference to the biggest set in networks of ten species.

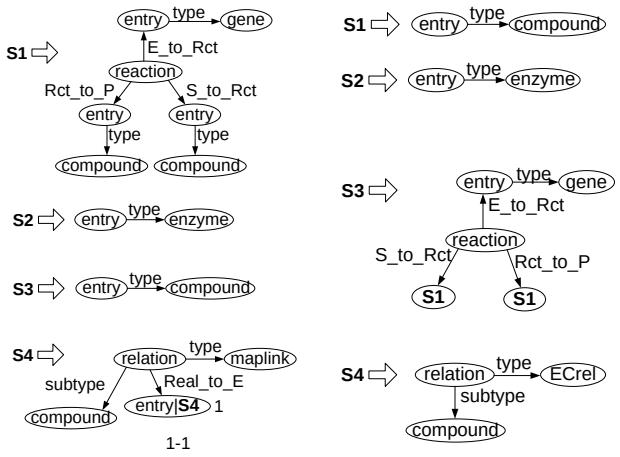

Figure 10: Graph grammar inferred from a set of thirty (a) and one hundred and ten (b) graphs of Picrophilus torridus (pto).

We would like to compare our inferred grammar from sets of different sizes to the original, true, ideal grammar which represents the species. However, such a graph grammar is not known. In the first experiment we adopted as an original grammar the grammar inferred from the last set. From each set we infer four grammar productions which score the highest in the evaluation. We compute the error (distance) of an inferred grammar to the grammar inferred from the set with all networks. The computation of an error is the same as it is described in section 5.4 on Learning Curves. The error is the minimal number of edges, vertices, and labels required to be change or removed to transform the structure of graph productions from one grammar to the other. In figures we refer to it as #transformations. In Figure 9 we show the results of the experiment. Every value in the Figure 9 is an average from three runs. In every run we randomly shuffle the networks over 11 sets such that sets are different in every run. In Figure 10 we show the graph grammar inferred from a set of thirty and a set of one hundred and ten graphs of Picrophilus torridus (pto).

The experiments on the biological network domain give us insight into the performance of the algorithm and to the biological networks. Examining Figure 9 we notice that some species, like dme, have a very regular set of biological networks. Increasing the size of the set does not change the inferred grammar. While in other species, like xcc, the set of biological networks is very diverse resulting in significant changes on the curve. Several curves, pto, pho, efa, gradually decrease with the last values being zero. It shows us that our algorithm performed well and with increasing number of graphs in the input set we find the grammar which does not change more with increased number of graphs which indicates that grammar found represents the input set well. The very bottom chart in Figure 9 shows the average change. We see that with the increasing number of graphs in the input sets the curve declines to zero which tells us that with the increasing number of graphs we infer more accurate grammar.

6. Conclusions and Future Work

We have studied algorithms for inferring node and edge replacement graph grammars. The algorithm starts from all nodes with the same label and grows them by adding to them one node or a node and an edge at a time. We developed a substructure which consists of the definition of a graph and all subgraphs appearing in the input graph that are isomorphic to this graph definition (i.e., instances). The overlap of instances proposes a recursive graph grammar production which expresses concepts of 'one or more' of the same substructures. The input graph to our algorithm is an arbitrary directed or undirected graph with labels on nodes and edges.

Grammar productions with graphs of higher complexity measured by MDL are inferred with smaller error. The error of grammar inference increases as the number of different labels used in the grammar decreases. In experiments with biological networks we notice that some species, like dme, have a very regular set of biological networks. Increasing the size of the set does not change the inferred grammar. While in other species, like xcc, the set of biological networks is very diverse. Several curves (pto, pho, efa), which represent the change in error with the increased sample set, gradually decrease, with the last values being zero. It shows us that our algorithm performed well and with an increasing number of graphs

in the input set we find the grammar, which does not change more with an increased number of graphs, which indicates that the grammar found represents the input set well.

Grammars inferred by the approach developed by Jonyer et al. (2004) were limited to chains of isomorphic subgraphs which must be connected by a single edge. Since the connecting edge can be included in the production's subgraph, and isomorphic subgraphs will overlap by one vertex, our approach can infer Jonyer et al.'s class of grammars..

We would like to indicate general future directions in graph grammar inference research. They are:

(1) Develop algorithms which allow for learning larger classes of graph grammars. We extended classes of presently learnable graph grammars. It is possible to extend it even further into context sensitive graph grammars where we could still replace nodes and edges, but whether or not this replacement takes place depends on the neighborhood of the replaced node or edge. In order to regenerate structures we would need more sophisticated generation mechanism with a context sensitive embedding mechanism. This mechanism, inferred during induction, would indicate nodes to merge during the generation process. We can explore other techniques like decomposition of graphs in searching for the best grammar which describes the data.

(2) Investigate learnable properties of graphs from the perspective of graph grammars.

(3) Identify experimental areas and show the significance of graph grammar inference in these domains. One of the new domains we approach is visual languages, where graph grammar inference from the sample of a language can give a grammar to be used to check newly written programs.

(4) Use graph grammar inference to identify building blocks, modularity and motifs in biology, software, social networks, and electronics circuits. We did experiments in biology and XML domains. Biological and chemical structures are still very promising areas of the application of recursive graph grammars. Social networks, Very Large Scale Integrated circuits, and the Internet are domains with relational data whose hierarchy and recursive properties we can explore with graph grammars.

(5) Expand graph grammar inference to learning stochastic graph grammars. This extension would require assigning a probability to each production. We can evaluate this probability based on the portion of the input graph covered by the inferred production.

References

Chomsky, N. (1956). Three models of language. IRE *Transactions in Information Theory* 2, 3, 113-24

Cook, D. and Holder L. (1994) Substructure Discovery Using Minimum Description Length and Background Knowledge. *Journal of Artificial Intelligence Research*, 1, 231-255.

Cook, D. and Holder, L. (2000). Graph-Based Data Mining. *IEEE Intelligent Systems*, 15(2), 32-41.

Jeltsch, E. and Kreowski, H. (1990). Grammatical Inference Based on Hyperedge Replacement. Graph-Grammars. *Lecture Notes in Computer Science* 532, 461-474.

Jonyer, I., Holder, L., and Cook, C. (2002) Concept Formation Using Graph Grammars, *Proceedings of the KDD Workshop on Multi-Relational Data Mining*.

Jonyer, I. Holder, L, and. Cook, D, (2004) MDL-Based Context-Free Graph Grammar Induction and Applications. International *Journal of Artificial Intelligence Tools*, Volume 13, No. 1 65-79.

Kanehisa, M., Goto, S., Hattori, M., Aoki-Kinoshita, K.F., Itoh, M., Kawashima, S., Katayama, T., Araki, M., and Hirakawa, M. (2006). From genomics to chemical genomics: new developments in KEGG. *Nucleic Acids Res*. 34, D354-357.

Kukluk, J., Holder, L., and Cook, D.(2006) Inference of Node Replacement Recursive Graph Grammars, *Sixth SIAM International Conference on Data Mining*

Kuramochi, M. and Karypis, G. (2001). Frequent subgraph discovery. In Proceedings of IEEE 2001 *International Conference on Data Mining* (ICDM '01), 313-320.

Neidle, S. (1999) (editor) Oxford *Handbook of Nucleic Acid Structure*. Oxford University Press, 326.

Nevill-Manning, G. and Witten, H. (1997). Identifying Hierarchical Structure in Sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, Vol 7, (1997), 67-82

Phan A.,. Kuryavyi V., Ma J, Faure A, Andreola M, Patel D. (2005) An interlocked dimeric parallel-stranded DNA quadruplex: A potent inhibitor of HIV-1 integrase. *Proceedings of the National Academy of Sciences*, 102, 634 – 639.

Oates T., Doshi S., and Huang F. (2003). Estimating Maximum Likelihood Parameters for Stochastic Context-Free Graph Grammars. *Lecture Notes in Artificial Intelligence*, 2835, 281-298. Springer-Verlag,

Rissanen, J. (1989). Stochastic Complexity in StatisticalInquiry. World Scientific Company.

Yan X. and Han J., gSpan (2002): Graph-based substructure pattern mining. In IEEE *International Conference on Data Mining*, Maebashi City, Japan.