

Inference of Node Replacement Recursive Graph Grammars

Jacek P. Kukluk, Lawrence B. Holder, and Diane J. Cook

{kukluk, holder, cook} @cse.uta.edu
Department of Computer Science and Engineering
University of Texas at Arlington
Box 19015, Arlington, TX 76019

Abstract

In this paper we describe an approach to learning node replacement graph grammars. This approach is based on previous research in frequent isomorphic subgraphs discovery. We extend the search for frequent subgraphs by checking for overlap among the instances of the subgraphs in the input graph. If subgraphs overlap by one node we propose a node replacement grammar production. We also can infer a hierarchy of productions by compressing portions of a graph described by a production and then infer new productions on the compressed graph. We validate this approach in experiments where we generate graphs from known grammars and measure how well our system infers the original grammar from the generated graph.

Keywords: Grammar Induction, Graph Grammars, Graph Mining.

1. Introduction

String grammars are fundamental to linguistics and computer science. Graph grammars can represent relations in data which strings cannot. Graph grammars can represent hierarchical structures in data and generalize knowledge in graph domains. In this paper we study the problem of grammar inference. We introduce an algorithm which builds on previous work in discovering frequent subgraphs in a graph [Cook94]. We check if subgraphs overlap and if they overlap by one node, we use this node and subgraph structure to propose a node replacement graph grammar.

We found only a few studies in graph grammar inference. Jeltsch and Kreowski [Jeltsch90] did a theoretical study of inferring hyperedge replacement graph grammars. Oates et al. [Oates03] discuss the problem of inferring probabilities of every grammar rule for stochastic hyperedge replacement context free graph grammars. In terms of similarity to string grammar inference we consider the Sequitur system developed by Nevill-Manning and Witten [Nevill97].

Sequitur infers hierarchical structure by replacing substrings based on grammar rules.

The most relevant work to this research is Jonyer et al.'s approach [Jonyer04]. Their system starts by finding frequently occurring subgraphs in the input graphs. Frequent subgraphs are those that when replaced by single nodes minimize the description length of the graph. They check if isomorphic instances of the subgraphs that minimize the measure are connected by one edge. If they are, a production $S \rightarrow PS$ is proposed, where P is the frequent subgraph. P and S are connected by one edge. Our approach is similar to Jonyer's in that we also start by finding frequently occurring subgraphs, but we test if the instances of the subgraphs overlap by one node. Jonyer's method of testing if subgraphs are adjacent by one edge limits his grammars to description of "chains" of isomorphic subgraphs connected by one edge.

2. Node replacement recursive graph grammar

We define a graph with 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. It 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, $\mu: V \rightarrow L$ - is a function assigning labels to the nodes, $\nu: E \rightarrow L$ - is a function assigning labels to the edges, $\eta: E \rightarrow \{0, 1\}$ - 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, and there are two types of productions:

(1) *recursive productions* of the form (d, G, C) , with $d \in \Sigma - \Delta$, G is a graph, where there is at least one node in G labeled d . 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

(1) *non-recursive production*, there is no node in G labeled d (our inference system does not infer an embedding mechanism for these productions).

3. The algorithm

An example in Figure 1 shows a graph composed of three overlapping substructures. The algorithm generates candidate substructures and evaluates them using the following measure of compression,

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

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. $size(g)$ can be computed simply by summing the number of nodes and edges: $size(g) = vertices(g) + edges(g)$. Another successful measure of $size(g)$ is the Minimum Description Length (MDL) discussed in detail in [Cook94]. Either of these measures can be used to guide the search and determine the best graph

grammar. In our experiments we used only the size measure.

The input to our algorithm is a graph G which can be one connected graph or set of disconnected graphs. G can have directed edges or undirected edges. The algorithm assumes labels on nodes and edges. The algorithm processes the list of substructures Q . In Figure 2 we see an example of a substructure definition. A substructure consists of a graph definition and a set of instances from the input graph that are isomorphic to the graph definition.

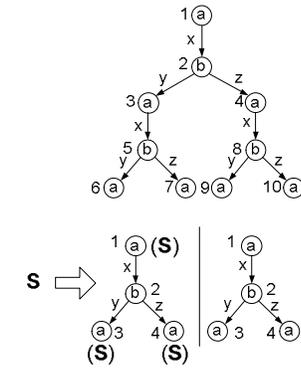


Figure 1: A graph with overlapping substructures and a graph grammar representation of it.

The algorithm starts with a list of substructures where every substructure is a single node and its instances are all nodes in the graph with this node label. The best substructure is initially the first substructure in the Q list. We extend each substructure 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 allow instances to grow and overlap, but any two instances can overlap by only one node. We keep all extended substructures in $newQ$. We evaluate substructures in $newQ$. The recursive substructure is evaluated along with non-recursive substructures and is competing with non-recursive substructures. The total number of substructures considered is determined by the input parameter *Limit*. We compress G with best substructure. Compression replaces every instance of best substructure with a single node. This node is labeled with a non-terminal label. The compressed graph is further processed until it cannot be compressed any more. In consecutive iterations 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 a beam search, i.e., the length of Q . For more details about the algorithm see [Cook94, Jonyer02, Jonyer04].

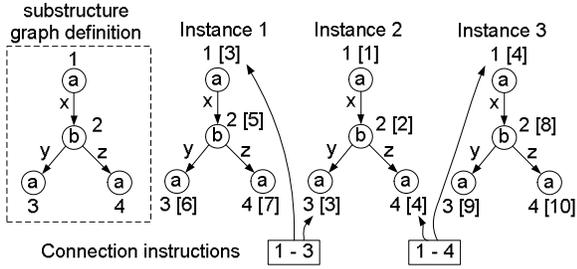


Figure 2: Substructure and its instances while determining connection instructions (continuation of the example from Figure 1).

4. Hierarchy of productions

In our first example from Figure 1, we described a grammar with only one production. Now we would like to introduce a complex example to illustrate the inference of a grammar which describes a more general tree structure. In Figure 3 we have a tree with all nodes having the same label. There are two repetitive subgraphs in the tree. One has three edges labeled “a,” “b,” and “c.” The other has two edges with labels “x” and “y.” There are also three edges K1, K2, and K3 which are not part of any repetitive subgraph. In the first iteration we find grammar production S1. The compressed graph, at this point, contains the node S1, edges K1, K2, K3 and subgraphs with edges “x” and “y.” In the second iteration our program proposes production S2. Compressing the tree with production S2 results in a graph which we use as an initial production S.

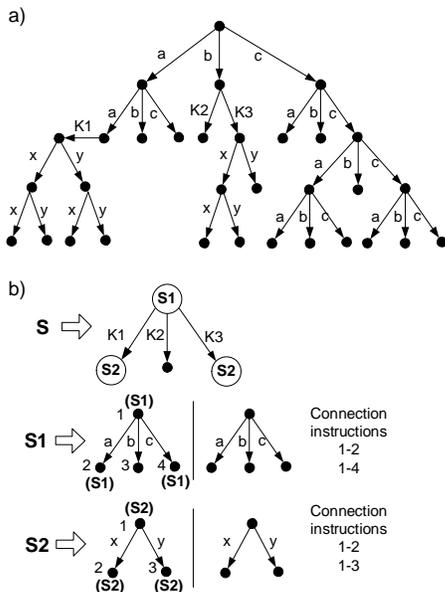


Figure 3: The tree (a) and inferred tree grammar (b).

5. Experiments

5.1. MDL as a measure of complexity of a grammar

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. The MDL measure, which while not provably minimal, is designed to be a near-minimal encoding of a graph. See [Cook94] for a more detailed discussion.

5.2. Error

We introduce a measure to compare the original grammar to the inferred 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

5.3. Experiment 1: Error as a function of noise and complexity of a grammar

We used twenty nine graphs which are all connected graphs with one, two, three, four and five nodes 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 $(\text{noise} / (1 - \text{noise})) * \text{number_of_nodes}$. Similar for edges. For every value of noise and MDL we generated thirty graphs and take average value of the error. We see trends in the plots in Figure 4. Error decreases as MDL increases. A low value of MDL is associated with small graphs, with three or four nodes and a few edges. These graphs, when used on the right hand side of a grammar production, generate graphs with fewer labels than grammars with high MDL.

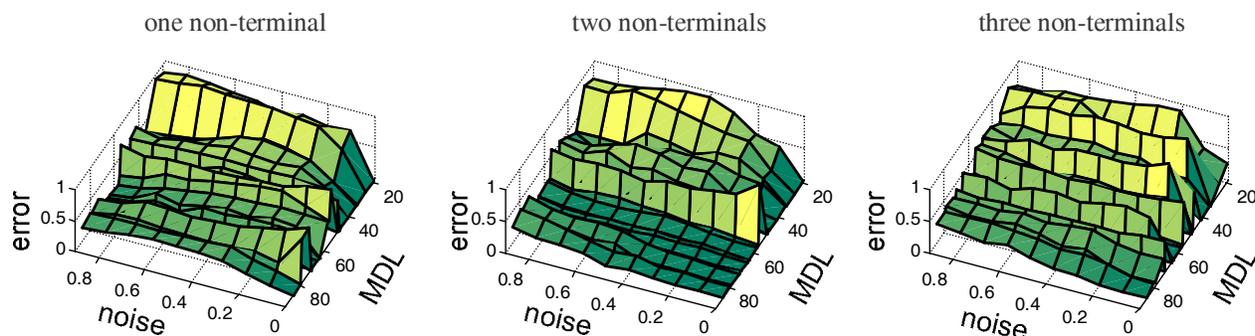


Figure 4: Error as a function of noise and MDL where graph structure was not corrupted.

Smaller numbers of labels in the graph increase the inference error, because everything in the graph looks similar, and the approach is more likely to propose another grammar which is very different than the original. One error occurs when inferred graph structure contains two overlapping copies of the graph used in the original grammar production. The structure has significant error, yet does subjectively capture the recursive structure of the original grammar.

5.4. Experiment 2: Error as a function of number of labels and complexity of a grammar

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 5. 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.

The average error for graphs with only one label is 1 or very close to 1. The most frequent inference error results from the tendency of our algorithm to propose one-edge grammars when inferred from a graph with only one label. We illustrate this in Figure 6 where we see a production with a pentagon using only one label “a”.

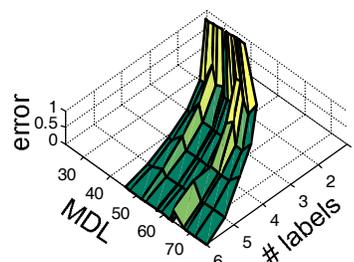


Figure 5 Error as a function of MDL and number of different labels used in a grammar definition (two non-terminals).

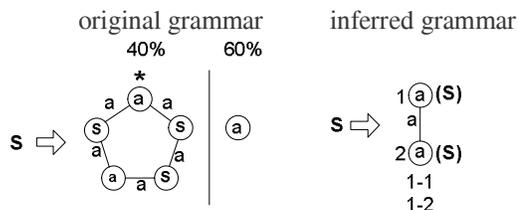


Figure 6: Error where inferred grammar is reduced to production with single edge.

5.5. Experiment 3: Chemical structure

As an example from the real-world domain of chemistry, we use the structure of cellulose with hydrogen bonding as the input graph in our next experiment. Figure 7 shows the structure of the molecule and the grammar production we found in this structure. The grammar production we found captures the underlying motif of the chemical structure. It shows the repetitive connected component, the basic building block of the structure. We can search for such underlying building motifs in different domains, hoping that they will improve our understanding of chemical, biological, computer, and social networks.

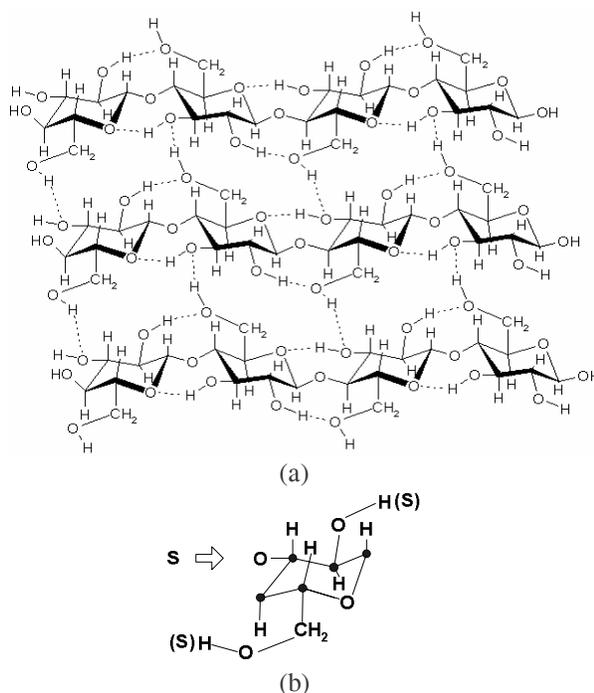


Figure 7: The structure of cellulose with hydrogen bonding (a) and the inferred grammar production (b).

6. Conclusion and future work

We described an algorithm for inferring certain types of graph grammars we call recursive node replacement graph grammars. The algorithm is based on previous work in frequent substructure discovery. It checks if frequent subgraphs overlap by a node and proposes a graph grammar if they do. The algorithm we described has its limitations: the left side of the production is limited to one single node; only connecting two single nodes is allowed in derivations. The algorithm finds recursive productions if repetitive patterns occur within an input graph and they overlap. If such patterns do not exist, the algorithm finds non-recursive productions and builds hierarchical structure of the input data. Grammar productions with graphs of higher complexity measured by MDL are inferred with smaller error. There is little dependency of error on noise if the generated graphs are not corrupted. The error of grammar inference increases as the number of different labels used in the grammar decreases. There is no dependency between the size of a sample graph and inference error. If all labels on nodes are the same and all labels on edges are the same, the algorithm produces a grammar which has only one edge in the graph definition. One-edge grammars over-generalize if the input graph is a tree, and they are inaccurate in many other graphs. This tendency to

find one-edge grammars from large, connected graphs is due to the fact that one-edge grammars score high because they can compress the graph well.

Grammars inferred by the approach developed by Jonyer et al. [Jonyer04] 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 noticed in our experiments that when the subgraphs overlap by more than one node, our algorithm still looks for overlap on only one node and infers a grammar which cannot generate the input graph. Therefore, one extension to the algorithm would be a modification which would allow for overlap larger than a single node.

References

- [Cook00] D. Cook and L. Holder, *Graph-Based Data Mining*. IEEE Intelligent Systems, 15(2), pages 32-41, 2000.
- [Cook94] D. Cook and L. Holder, *Substructure Discovery Using Minimum Description Length and Background Knowledge*. Journal of Artificial Intelligence Research, Vol 1, (1994), 231-255
- [Doshi02] S. Doshi, F. Huang, and T. Oates, *Inferring the Structure of Graph Grammar from Data*. Proceedings of the International Conference on Knowledge Based Computer Systems (KBCS'02)
- [Jeltsch90] E. Jeltsch, H. Kreowski, *Grammatical Inference Based on Hyperedge Replacement*. *Graph-Grammars*. Lecture Notes in Computer Science 532, 1990: 461-474
- [Jonyer04] I. Jonyer and L. Holder, and D. Cook, *MDL-Based Context-Free Graph Grammar Induction and Applications*. International Journal of Artificial Intelligence Tools, Volume 13, No. 1 pages 65-79, 2004.
- [Nevill97] G. Nevill-Manning and H. Witten, *Identifying Hierarchical Structure in Sequences: A linear-time algorithm*. Journal of Artificial Intelligence Research, Vol 7, (1997), 67-82
- [Oates03] T. Oates, S. Doshi, and F. Huang, *Estimating Maximum Likelihood Parameters for Stochastic Context-Free Graph Grammars*. Volume 2835 of Lecture Notes in Artificial Intelligence, pages 281-298. Springer-Verlag, 2003.