Graph Grammar Induction on Structural Data for Visual Programming

Keven Ates[1], Jacek Kukluk[2], Lawrence Holder[2], Diane Cook[2], Kang Zhang[1]
[1]*University of Texas at Dallas,* [2]*University of Texas at Arlington*
atescomp@utd.edu, kukluk@uta.edu, holder@uta.edu, cook@uta.edu, kzhang@utd.edu

Abstract

Computer programs that can be expressed in two or more dimensions are typically called visual programs. The underlying theories of visual programming languages involve graph grammars. As graph grammars are usually constructed manually, construction can be a time-consuming process that demands technical knowledge. Therefore, a technique for automatically constructing graph grammars—at least in part—is desirable. An induction method is given to infer node replacement graph grammars. The method operates on labeled graphs of broad applicability. It is evaluated by its performance on inferring graph grammars from various structural representations. The correctness of an inferred grammar is verified by parsing graphs not present in the training set.

1. Introduction

Visual programming languages (VPLs) provide the means by which programs are specified and executed in two or more dimensions. The underlying theories of VPLs include graph grammars. Graph grammars assist the creation, editing, analysis, and execution of visual programs. Many environments are easily expressed in graph forms. For example, textual computer programs or data can be represented as tree structures. Also, diagram specifications include the UML, Petri nets, database designs, control flow programming, and state transitions [6]. Labeled graphs with nodes as entities and edges as relations between nodes are a good abstraction of such diagrams [5], which can be expressed as visual programs.

Generally, graph grammars are difficult to construct and require considerable expertise. This provides a motivation to automate the graph grammar construction process. Using an induction engine is one solution that can simplify this construction process. Sample instances of a language are often available. As a training set, these samples are processed quickly and automatically to construct a graph grammar. The induced graph grammars can be further modified. The main idea is illustrated in Figure 1. Additionally, the resulting graph grammar can be verified for correctness by parsing graphs of the language outside the training set. Furthermore, a grammar induction system paired with a parsing system has some interesting implications for the management of grammars and its impact on machine learning.

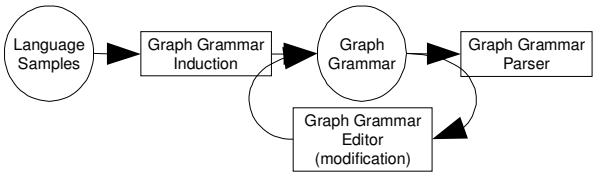

Figure 1: System for identifying graph grammars from language samples

The correctness of the specification for a graph form can be controlled using graph grammars. The inference algorithm described in Section 4 is used to infer node replacement graph grammars. It operates on labeled graphs of broad applicability. Two applications for this method are shown. One application infers a graph grammar from a tree representation of a partial computer program. Another infers a graph grammar from a structural representation of XML data. The correctness of this second grammar is verified by parsing both valid and invalid graphs not found in the training set.

A graph is defined with labeled nodes and edges. Every edge of the graph can be directed or undirected. The definition of a graph grammar is described as the class of grammars that can be inferred by the induction method, which are currently limited to context-free grammars. The main characteristic of the inferred grammar productions is that they are recursive productions. Recursive productions are inferred such that, for a given production, a non-terminal node label on the left side appears one or more times in the node labels of the graph on the right side. The method can also infer non-recursive productions which are frequent, non-overlapping subgraphs of an input graph. The embedding

Proceedings of the 18th IEEE International
Conference on Tools with Artificial Intelligence (ICTAI'06)
0-7695-2728-0/06 $20.00 © 2006 IEEE

mechanism of the grammar consists of connection instructions. Every connection instruction is a node pair that indicates where the production graph can recursively connect to itself.

A review of existing parsing and inference approaches is presented. Then, the proposed inference algorithm is presented. Two experiments using the inference algorithm are examined. The first uses a partial computer program. The second uses an XML file for an example book order system and demonstrates parsing results that examine the correctness of the inferred graph grammar. The conclusion follows discussing our results and describing future work in visual programming and machine learning using the combined induction-parsing system.

2. Graph grammar parsing systems

The Layered Graph Grammar (LGG) environment [14] was an important early system that provided for the construction of context-sensitive graph grammars. LGG was motivated by need for visual languages to have more powerfully expressive grammars than context-free graph grammars could provide. This led to the development of a layered approach where label elements are assigned within the grammar to produce a more "fine-grained decomposition" than the previously accepted non-terminal and terminal sets provided. As in most context-sensitive parsing, the pure implementation of LGG contained an exponential time algorithm. Other improvements have reduced the complexity to polynomial time algorithms.

The VisPro environment [18] was a next generation system that extended and simplified the context-sensitivity of a grammar by introducing the Reserved Graph Grammar (RGG) [17] which used embedding rules combined with context elements in the construction of a grammar. The context elements provided structural relations that helped direct the parsing process. VisPro improved upon the exponential time parsing process by introducing a constraint that required the grammars to be confluent, resulting in a polynomial time algorithm. RGG's simplification over LGG allowed for the expanded use of graph grammars into many disciplines.

An extension to RGG, RGG+ [16] introduced a size increasing condition on the production rules which simplified the grammar definition to traditional non-terminal and terminal elements. It also provided for non-confluent grammars. This expanded a grammar's expressive power, but resulted in an alternate exponential time algorithm.

As a more important extension to RGG, the Spatial Graph Grammar (SGG) [9] system added spatial specifications to the production rules in RGG grammars. This allowed spatial constraints between elements to influence the parsing process. Production rules, identical in all other ways, are allowed to differ by their nodes' spatial relationships. Spatial specifications attached to a rule determine when the spatial relationships apply. As an example application, SGG has been used in the parsing of web pages for the purpose of restructuring its presentation on alternate viewing devices such as PDAs and cellular phones [10]. As one of the latest tools in the graph grammar world, SGG is used in the experiments shown in this paper.

3. Graph grammar induction systems

The development of induction systems has a rich and varied background. However, the vast majority of induction systems are developed for text based (or one dimensional) data. In contrast to text based induction systems, the development of graph grammar induction systems has been sparse. One of the earliest examples for the induction of graph grammars is a process that uses an enumerative technique to infer a limited class of context-sensitive graph grammars [4]. However, there have been more recent developments in inference systems for graph grammars.

As an important modern advancement for graph grammar induction, the "association rule discovery" approach [2] was developed. This resulted in an algorithm, Apriori [3], which is used to discover "frequent itemsets" within a graph. This algorithm has resulted in a range of applications such as Apriori-based Graph Mining (AGM) [7], FSG [13], and gSpan [15]. However, Apriori based systems are generally limited to a single connected graph as the training set.

Another modern graph grammar induction system, SubdueGL [8], has also been developed. As a competing system, SubdueGL uses an MDL based heuristic approach. A significant benefit over Apriori is that it is able to accept multiple graphs in its training set. It is also able to accept arbitrary graphs having labeled nodes and edges. Any data that can be represented in graph form is suitable for the training set. SubdueGL infers the structural similarities through the MDL principle to produce a common context-free graph grammar.

The basis for most induction methods is firmly rooted in inductive logic programming (ILP) in which the data is represented using first-order logic. An induction process is performed on the logic to produce a learned set, or generalization, of rules on the data. The expressive power of ILP algorithms is ideal since they result in recursive expressions with variable and don't-care terms [8]. As an alternative, some methods use the genetic programming (GP) approach which

Proceedings of the 18th IEEE International
Conference on Tools with Artificial Intelligence (ICTAI'06)
0-7695-2728-0/06 $20.00 © 2006 IEEE

has had limited success. It is suggested that the GP approach is ideal due to the symbolic nature of grammars and their tree-like representation [11]. However, the recursive nature of grammars has limited its usefulness.

4. Inference algorithm

The following describes a new algorithm for node replacement graph grammar inference based on overlapping subgraphs [12]. The input for the algorithm is a training set of labeled arbitrary graphs. The algorithm starts by finding all nodes with the same label and storing each of them as initial subgraphs in a list called Q. Each iteration creates new candidate subgraphs by expanding all the subgraphs in the list Q by one edge or an edge and a node. These are kept in a list called *newQ*. These candidates for the so-called "best subgraphs" are evaluated as follows. Every occurrence of a candidate subgraph within the entire graph is temporarily replaced by a new node. The compression achieved with this replacement is measured by calculating a minimum description length or size (number of nodes + number of edges) of the original and compressed graph. The subgraphs in *newQ* with the highest compression ratio are the "best subgraphs". These become the new list Q. The following example elaborates on the process described above.

A graph composed of three overlapping subgraphs is shown in the top half of Figure 2 and its graph grammar representation is shown in the bottom half of Figure 2. The algorithm generates candidate subgraphs and evaluates them using the following measure of compression,

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

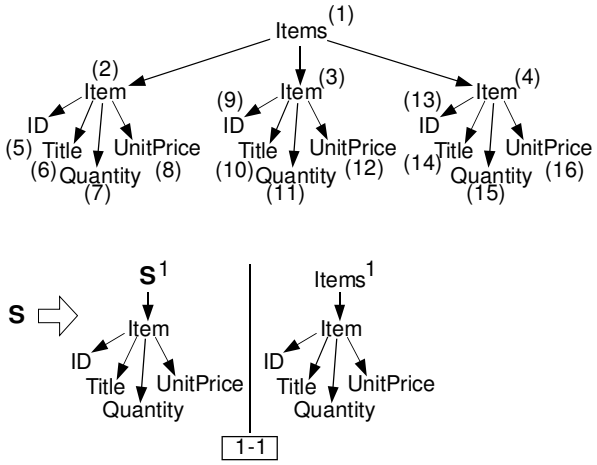

Figure 2: A graph with overlapping subgraphs and its graph grammar representation

where G is the input graph, S is a subgraph, and $G|S$ is the graph derived from G by compressing each instance of S into a single node. The $size\ (G)$ element 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) [5]. Either of these measures can be used to guide the search and determine the best graph grammar.

Instances are allowed to grow and overlap with the restriction that the overlap is limited to only one node. The substructure found in the graph of the top half of Figure 2 is shown in the top half of Figure 3. The "Items" node is the one node that is allowed to overlap. The bottom half of Figure 3 shows each instance of the substructure where the arrows indicate how each instance overlaps with the others on the "Items" node. The "(1)" indicates the graph id for the node and the "1" indicates the substructure id of the node in both figures.

The candidate subgraphs in the list *newQ* are evaluated for the compression process. Therefore, *newQ* is composed of subgraphs that are recursive/overlapping and non-recursive/non-overlapping. Each candidate competes with all others. To further control the process, the input parameter *Beam* specifies the width of the beam search for the best subgraphs in *newQ* that become the new list stored in Q, i.e. the length of Q [12]. The candidate process is then repeated. The total number of subgraphs considered is determined by the input parameter *Limit*. G is then compressed with the best subgraph found over all iterations of the process. The compression process replaces every instance of the best subgraph with a single node. This node is labeled with a non-terminal label. The compressed graph is

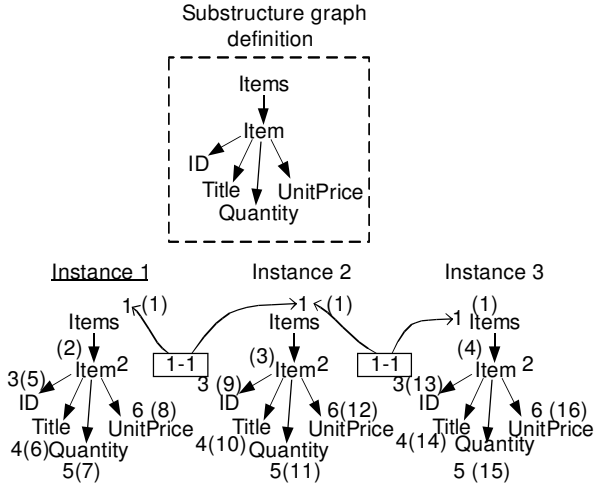

Figure 3: Subgraph and instances determining connection instructions (continues Figure 2)

Proceedings of the 18th IEEE International
Conference on Tools with Artificial Intelligence (ICTAI'06)
0-7695-2728-0/06 $20.00 © 2006 IEEE

repeatedly processed for candidates and compression until it cannot be compressed further. In consecutive iterations, the best subgraph can have one or more non-terminal labels. The algorithm results in a hierarchy of grammar productions where each compression produces a rule.

A prototype system was developed to test the inference algorithm. The system consists of a program implementing the induction algorithm and the existing SGG editor/parser software. A training set of graphs are input to the induction algorithm producing the hierarchy of grammar productions. The grammar productions are then manually entered into the SGG software by identifying all terminal and non-terminal nodes, constructing the given production rules, and generating the required graphs. The grammar is then used to validate graphs within the graph domain both within and outside the training set.

5. Experiments

Two experiments are presented to validate the inference system. The first experiment is a rapid grammar development example showing the grammar inferred from the graph of a parse tree for a partial text program. The second experiment is a machine learning example showing the inferred graph grammar from an XML file containing sample book order data. This last experiment verifies the "learned knowledge" of the inferred grammar by processing a set of domain graphs with the SGG parser.

In order to use XML files, a simple converter was developed to convert the files into trees. The Java implementation of Document Object Model (DOM) is used in the converter. According to [1] there are twelve DOM node types: Element, Attr, Text, CDATASection, EntityReference, Entity, ProcessingInstruction, Comment, Document, DocumentType, DocumentFragment, and Notation. However, the converter was designed to build a directed tree with the root node always labeled DOC and then only extract the 'Element' data types in the experiments. From the perspective of our graph grammar inference system, it requires a pattern which is repeated in the graph, so the converter eliminates unique text data (e.g., names, card numbers, and price values) which is consider node attribute data. Labels are not assigned to the tree edges.

```
Program MyProg
{
    Class A
    {
       int main()
       {
          int A;
          string B;
       }
       int Meth2()
       {
          string CC;
          int DA;
       }
    }
    Class B
    {
       string Meth1()
       {
          int DA;
          int DB;
       }
       A Meth2(string YY) //return Class A
object
       {
          string XX;
       }
    }
```

Figure 4: Partial text program

5.1. Inferencing on a tree representing a program

A partial program is shown in Figure 4 which was converted to the graph shown in Figure 5. The program is partial by design with the purpose of eliminating statements and detailed formal field declarations. Statements are removed due to graph complexities on the variety of statement types. They would normally appear in Statement Lists in a final "BodyOp" in a "MethodOp". Field declarations are minimized due to graph complexity such as initialization expressions, array creation, and multiple variables on a single type. They would normally have complex recursive "CommaOp" subgraphs connected to recursive "DecOp" nodes. The "DecOp" nodes are now singleton declarations connected to individual "BodyOp" nodes for a class or method. This simplifies the construction of a graph and the analysis of its inferred graph grammar.

The upper graph of Figure 5 represents a VPL mapping of the text program where the node labels have specific meanings. Branch nodes are labeled with "Op" and leaf nodes are labeled with "Node". "ProgramOp" means program, the root node operator. "BodyOp" means class or method body. "DeclOp" means declaration operator. "CommaOp" means list separator. "HeadOp" means head of method. "SpecOp" means method parameters, operator. "ClassOp" means head of class. "ClassDefOp" means class definition operator. "MethOp" means method operator. "TypeNode" means type identifier. "IdNode" means attribute identification. Each node represents some subsection of the program. For example, "Program MyProg" is represented by a "ProgramOp" node connected to a "IdNode" node, where the "IdNode" contains the attribute "MyProg". For this experiment, the attribute data does not contribute to the inferred graph grammar and is not included in the figures.

The lower part of Figure 5 shows the inferred production rules. The first production, S, is the initial graph compressed with production S3. Production S

Proceedings of the 18th IEEE International
Conference on Tools with Artificial Intelligence (ICTAI'06)
0-7695-2728-0/06 $20.00 © 2006 IEEE

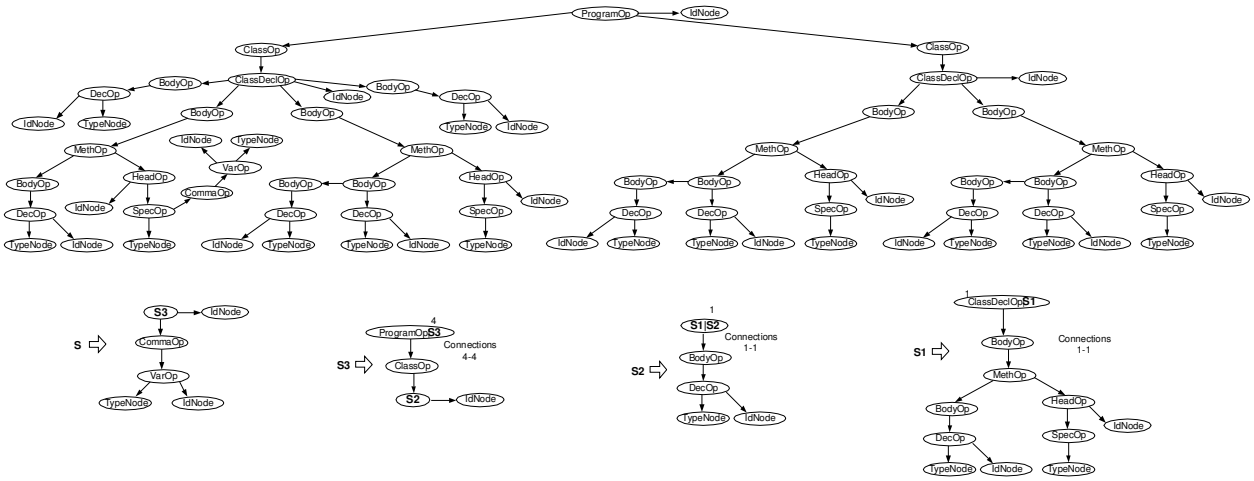

Figure 5: Graph of a partial text program and its inferred grammar productions

is non-recursive. Production S3 is recursive and demonstrates overlapping on a node with label "ClassOp", which conveys the idea that a program can have one or more classes. It is also expanded by production S2. Production S2 is recursive and shows that a class declaration can have one or more variable declarations. It is also expanded by production S1, which implements the "ClassDeclOp" node. Production S1 compresses the input graph the most, is recursive, and shows that a class declaration can have one or more methods.

Reviewing the grammar indicates that the induction process has limitations. Since the algorithm finds the best compressing subgraph on any iteration, the grammar may not be ideal. The "BodyOp" nodes under "MethOp" should have their own recursive production. However, since the S1 production is ideal by the algorithm, the chained "BodyOp" subgraphs are moved to the S2 production and are mixed with the "BodyOp" nodes of "ClassDefOp". The "CommaOp" node under "SpecOp" shows the same limitation where the "CommaOp" subgraph is moved to the S production. This was expected as the "CommaOp" subgraph is a singleton. Using additional graphs that better model these problem areas can guide the algorithm to create a better grammar.

These automated productions demonstrate the rapid development of a basic graph grammar that can be modified for domain specific purposes. Perhaps existing domain productions could be added and linked to inferred productions to create a new language. Providing a robust set of initial graphs can provide a developer with a "close-to-ideal" graph grammar.

5.2. Evaluating inferred graph grammar productions

To demonstrate the usefulness of a graph grammar induction process, this subsection presents an induction-parsing scenario for a graph grammar. First, the induction system, using the graph grammar induction algorithm described in Section 4, is used on sample graphs to produce a graph grammar. Second, the produced grammar is input to a graph grammar parsing system, an SGG derivative of VisPro, to test for correctness. Correctness is verified by parsing the original samples as well as derived graphs not processed by the induction system.

In Figure 6, an example XML data file is shown for a book order system. The graph representation of the file is shown in Figure 7. In Figure 8, the Document Type Definition (DTD) file that describes the XML files is shown for the book order system. Compared with the DTD file, the inferred grammar shown in Figure 9 demonstrates the accuracy of the induction system. The "ROOT" production shows the initial graph reduction indicating the entire DTD file. The "S1" productions show that the "Orders" node can have zero or more "Order" nodes as indicated by

```
<Orders>
<Order>
  <Customer>
    <Name>Bill Buckram</Name>
    <Cardnum>234 234 234 234</Cardnum>
  </Customer>
    <Receipt>
      <Subtotal>$53.75</Subtotal>
      <Tax>$4.43</Tax>
      <Total>$58.18</Total>
    </Receipt>
    <Items>
      <Item>
        <ID>209</ID>
        <Title>Duke: A Biography of the Java
               Evangelist
        </Title>
        <Quantity>1</Quantity>
        <UnitPrice>$10.75</UnitPrice>
      </Item>
...
</Order>
</Orders>
```

Figure 6: An XML file with book order data

Proceedings of the 18th IEEE International
Conference on Tools with Artificial Intelligence (ICTAI'06)
0-7695-2728-0/06 $20.00 © 2006 IEEE

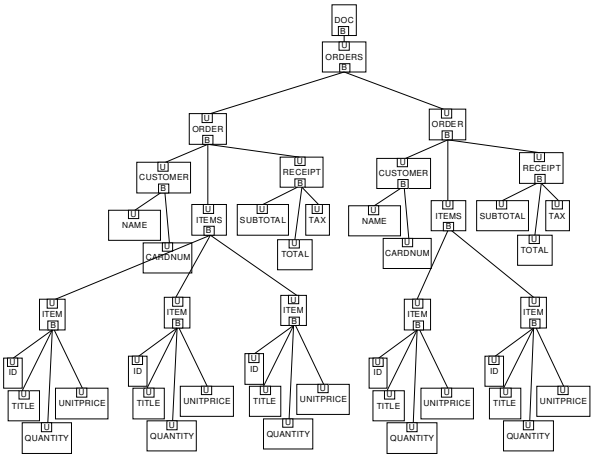

Figure 7: Graph of book order system XML file

the "<!ELEMENT Orders (Order*)>" line from the DTD file. The "S2" productions show that the "Items" node can have zero or more "Item" nodes as indicated by the "<!ELEMENT Items (Item*)>" line from the DTD file.

In Figure 7, Figure 9, and Figure 10, the use of node notation from the SGG system differs from the previous examples. The "B" and "U" vertices in each node are designated by the user and serve as connection placeholders used by the SGG system to convey context-sensitive information. In this example no context-sensitive information is conveyed by these vertices reducing the productions to a context-free application.

The original input graph was tested for correctness against the inferred grammar along with several other derived graphs. In Figure 10, one of the derived graphs is shown. Each graph either tests as a "Valid graph" or an "Invalid graph" by the parser. Graphs that successfully parse produce a parse tree indicating the production rules under which the parsing completed. Figure 11 is the parse tree representation of the graph from Figure 10. Graphs

```
<!ELEMENT Orders (Order*)>
<!ELEMENT Order (Customer,Items,Receipt)>
<!ATTLIST Order xmlns CDATA #FIXED
"http://www.example.com/myschema.xml">

<!ELEMENT Customer (Name, Cardnum)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Cardnum (#PCDATA)>

<!ELEMENT Items (Item*)>

<!ELEMENT Item (ID,Title,Quantity,UnitPrice)>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT UnitPrice (#PCDATA)>

<!ELEMENT Receipt (Subtotal,Tax,Total)>
<!ELEMENT Subtotal (#PCDATA)>
<!ELEMENT Tax (#PCDATA)>
<!ELEMENT Total (#PCDATA)>
```

Figure 8: DTD file for book order system

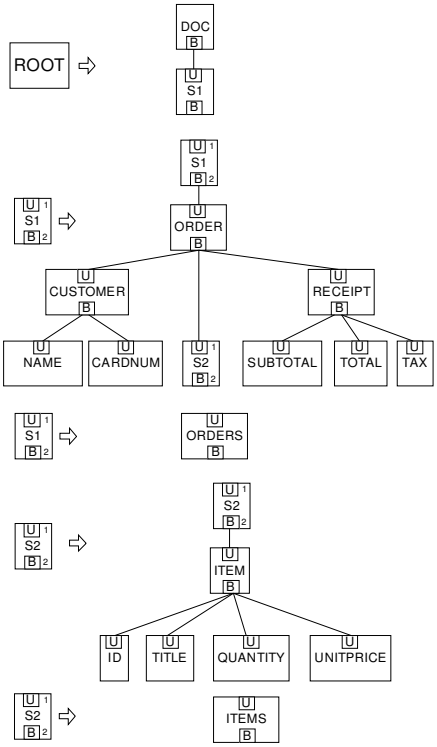

Figure 9: Inferred grammar of book order data

that fail to parse do not produce a valid parse tree—only the "Invalid graph" result is indicated.

A set of derived graphs were created to test the correctness of the inferred grammar using the SGG parser. Derived graphs were hand crafted derivatives of the original graph using additions and reductions. For example, Figure 10 has all item subtrees removed. The set consisted of a group of graphs that were known to satisfy the DTD and another group that were known to not satisfy the DTD. The original input graph, a satisfying graph, parsed successfully. Another satisfying graph consisting of two connected nodes, "DOC" and "Orders" parsed successfully. The satisfying graph of Figure 10 also parsed successfully. A non-satisfying graph that failed parsing was one that added an "Order" node without child nodes to an "Orders" node. Other non-satisfying graphs were

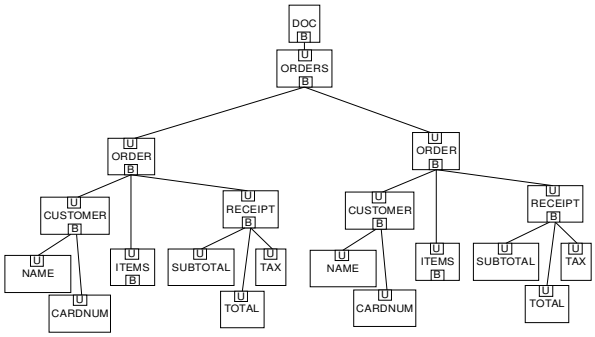

Figure 10: A graph used in the parsing process to verify an inferred graph grammar

Proceedings of the 18th IEEE International
Conference on Tools with Artificial Intelligence (ICTAI'06)
0-7695-2728-0/06 $20.00 © 2006 IEEE

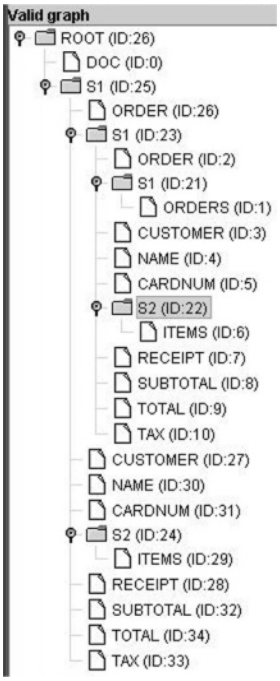

Figure 11: Parsing results of Figure 10 graph

ones that had random leaf nodes, such as "QUANTITY", "TAX", and "CARDNUM", removed from satisfying graphs like those in Figure 7 and Figure 10. Another was one that added an "Item" node without child nodes to an "Items" node.

The analysis showed that all graphs of the satisfying group parsed successfully and all graphs of the non-satisfying group failed with no exception. This was expected as the DTD file and the inferred grammar represent equivalent structures by inspection. However, the SGG parser has a requirement of confluence for a graph grammar [10]. Grammars generated by the induction algorithm are not guaranteed to be confluent and proof of confluence is thought to be NP-hard in general. Therefore, the success of the induction-parsing process only indicates its use in related applications.

6. Conclusion and future work

This paper has presented a graph grammar induction process using structural data that is applicable to visual programming. Section 4 described an algorithm for graph grammar induction. The presence of overlapping subgraphs in many graph domains proposes the induction of recursive graph grammar productions expressing the concept of "one or more" of the same subgraphs. The input graph to the algorithm is a training set of arbitrary directed or undirected graphs with labels on nodes and, optionally, edges.

In this paper, the algorithm is applied to trees with labels on nodes and directed unlabeled edges. The trees were generated from the structural representation of a computer program and XML files. The graph grammar inference algorithm was used to infer grammars from these trees. The partial program example demonstrated the application of the induction algorithm to rapid grammar development for VPLs. The book order system demonstrates that the graph grammar inference algorithm can extract the organization and hierarchy of the structure of XML files. Comparing the inferred graph grammar to the DTD indicated a strong correspondence between the DTD statements and graph grammar productions.

The graph grammar inference algorithm was created to process labeled arbitrary graphs. In this paper, it was applied only to tree graphs of a partial program and data stored in XML files. As such, the trees only partially demonstrate the power of the inference algorithm. Future plans are to apply the graph grammar induction algorithm to other graph domains such as networks, state transitions diagrams, and class diagrams.

An area of future research is context-sensitive graph grammars (CSGGs) and to study the induction algorithm's performance in rapidly developing VPLs. Inferring context sensitive productions in graph grammars is a goal for extending the algorithm. The expressive power of CSGGs gives adequate motivation to investigate the usefulness of such grammars in research. Other motivations are due to the development of CSGG construction and management tools, such as the SGG system. In general, complexities associated with graph grammar construction call for methods that reduce those complexities. An induction algorithm can mitigate such complexities by automating much of the design process for many difficult graph grammars. When accurate graph instances of a target grammar are used to create an inferred grammar, the initial development time can be drastically reduced. However, one concern is the issue of confluence as it has an impact on the time complexity of the parser. Generating graph grammars that have some guarantee of confluence is an additional goal of this work.

Another area of future research is in machine learning. Using existing technology, the pairing of an induction system with a parser was demonstrated. This pairing has interesting consequences for artificial intelligence (AI) applications in the Visual Computing domain. Creating an induction-parsing loop implements a basic machine learning system. The current state of a graph grammar may be modified by the induction system such that the parser validates a growing domain of sample graphs—the system's "knowledge" of some target domain increases. The

Proceedings of the 18th IEEE International
Conference on Tools with Artificial Intelligence (ICTAI'06)
0-7695-2728-0/06 $20.00 © 2006 IEEE

parsing process is a "truth engine" and the induction process changes the system's perception of truth.

The graph grammar induction-parsing system presented successfully extends the concept of the traditional text based machine learning system to the higher dimensional visual programming domain. Among the many applicable domains are web page restructuring and bioinformatics. A specific set of web pages requires a localized graph grammar to successfully restructure the pages to a given display device. Therefore, a "learn and use" process is required. Emerging bioinformatic research requires data mining and analysis of large and complex multi-dimensional data. Structures via a grammar can be learned from existing models through the induction process. Proposed models can be tested by parsing them against the known grammar of a model domain.

7. Acknowledgments

Jun Kong is acknowledged for his help with the SGG parsing system.

8. References

[1] K. Ahmed., S. Ancha, A. Cioroianu, J. Cousins, J. Crosbie, J. Davies, K. Gabhart, S. Gould, R. Laddad, S. Li, B. Macmillan, D. Rivers-Moore, J. Skubal, K. Watson, S. Williams, and J. Hart, 2001, *Professional Java XML*, WROX.

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

[3] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules", In *Proceedings of the International Conference on Very Large Databases*, pages 487-499, Santiago, Chile, 1994

[4] B. Bartsch-Spörl, "Grammatical inference of graph grammars for syntactic pattern recognition", *Lecture Notes in Computer Science*, 153: 1-7, 1983

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

[6] H. Ehrig, G. Engels, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation*, Volumes 1-3. World Scientific, 1997-1999

[7] A. Inokuchi, T. Washio and H. Motoda, "An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data", *Proceedings of the European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2000

[8] I. Jonyer, "Context-Free Graph Grammar Induction Based on the Minimum Description Length Principle," *Doctoral Dissertation*, The University of Texas at Arlington, August 2003

[9] J. Kong and K. Zhang, "On a Spatial Graph Grammar Formalism", *Proceedings of VL/HCC'04 - 2004 IEEE Symposium on Visual Languages and Human Centric Computing*, Rome, Italy, 26-29 September 2004, IEEE CS Press, pp. 102-104

[10] J. Kong, K. Zhang, and X. Q. Zeng, "Spatial Graph Grammars for Graphical User Interfaces", *ACM Transactions on Computer-Human Interaction*, 2006 (In Press)

[11] E. E. Korkmaz and G. Ucoluk, "Genetic Programming for Grammar Induction", *Proceedings of 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, 2001

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

[13] M. Kuramochi and G. Karypis, "An Efficient Algorithm for Discovering Frequent Subgraphs", *Technical Report 02-026*, Department of Computer Science, University of Minnesota, 2002

[14] J. Rekers and A. Schurr, "Defining and parsing Visual Languages with layered graph grammars". *Journal of Visual Languages and Computing*, 8(1):27-55, 1997

[15] X. Yan and J. Han, "gSpan: Graph-Based Substructure Pattern Mining", *Proceedings of the International Conference on Data Mining (ICDM)*, 2002

[16] X. Zeng, K. Zhang, J. Kong, and G-L Song, "RGG+: An Enhancement to the Reserved Graph Grammar Formalism", *VLHCC*, pp. 272-274, 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), 2005

[17] D. Zhang, K. Zhang, and J. Cao, *"A Context-sensitive Graph Grammar Formalism for the Specification of Visual Languages"*, The Computer Journal, 44(3), 2001, 186-200

[18] K. Zhang, D-Q. Zhang, and J. Cao, "Design, Construction, and Application of a Generic Visual Language Generation Environment", *IEEE Transactions on Software Engineering*, Vol.27, No.4, April 2001, 289-307