Graph Grammar Induction by Genetic Programming

Linting Xue
Electrical Engineering Department
North Carolina State University
Raleigh, North Carolina, U.S.A.
lxue3@ncsu.edu

Collin F.Lynch
Computer Science Department
North Carolina State University
Raleigh, North Carolina, U.S.A.
cflynch@ncsu.edu

Min Chi
Computer Science Department
North Carolina State University
Raleigh, North Carolina, U.S.A.
mchi@ncsu.edu

ABSTRACT

Augmented Graph Grammars provide a robust formalism for representing and evaluating graph structures. With the advent of robust graph libraries such as AGG, it has become possible to use graph grammars to analyze realistic data. Prior studies have shown that graph rules can be used to evaluate student work and to identify empirically-valid substructures using hand-authored rules. In this paper we describe proposed work on the automatic induction of graph grammars for student data using evolutionary computation via the pyEC system.

Keywords

Augmented Graph Grammars, Graph Data, Evolutionary Computation

1. INTRODUCTION

Graph Grammars are logical rule representations for graph structures. They can be designed to encode classes of suitable graphs to to recognize complex sub-features. They were introduced by Rosenfeld and Pfaltz in 1969 as "Context-free web grammars" [1]. Since then Graph grammars have been applied to a wide range of areas, including pattern recognition [2, 3, 4]; visual programming languages [5]; biological development [6]; classification of chemical compounds [7, 8]; and social network analysis [9, 10, 11]. Simple graph grammars are, like string grammars, composed of a set of production rules that map from one structure to another. In this case the rules map from a simple subgraph, typically a single node or arc, to a more complex structure. As with string grammars the node and arc types are drawn from finite alphabets. Despite their utility, however, graph grammars are very difficult to construct. The data structures required are complex [5]. Moreover, development of suitable graph grammars generally requires considerable domain expertise. Most existing uses of graph grammars have relied on hand-authored rules.

In this paper we describe our ongoing work on the automatic induction of Augmented Graph Grammars via Evolutionary Computation (EC). Our long-term goal in this work is to develop automated techniques that can extract empirically-valid graph rules which can, in turn, be used to classify student-produced argument diagrams and to provide the basis for automated student guidance and evaluation. This will build upon our prior on the evaluation of *a-priori* rules for student arguments. We will begin with background material on Augmented Graph Grammars and discuss prior work on grammar induction. We will then present an overview of our planned work.

2. AUGMENTED GRAPH GRAMMARS & ARGUMENT DIAGRAMS

Classical graph grammars are designed to deal with fixed graphs that are composed from a finite set of static node and arc types. Augmented graph grammars are an extension of simple graph grammars that allow for complex node and arc types, optional substructures, and complex rule expressions [12]. Rather than using a fixed alphabet of graph components they are defined by a complex ontology that allows for subsidiary types such as textual fields, access functions, and directional information. They can also be used to evaluate negated elements as well as quantified expressions. As such they are better suited to rich graph data such as user-system interaction logs and student-produced argument diagrams.

Augmented Graph Grammars have previously been used for the detection of empirically-valid substructures in student-produced argument diagrams [13, 14]. In that work *a-priori* rules were used to represent key discussion features and argumentative flaws. Argument diagrams are graphical argument representations that reify key features of arguments such as hypothesis statements, claims, and citations as nodes and the supporting, opposing, and informational relationships as arcs between them.

A sample diagram collected in that work is shown in Figure 1 The diagram includes a central research *claim* node, which has a single text field indicating the content of the research claim. A set of *citation* nodes are connected to the *claim* node via a set of *supporting, opposing* and *undefined* arcs colored with green, red and blue respectively. Each citation node contains two fields: one for the citation information, and the other for a summary of the cited work; each arc has a single text field explaining why the relationship holds. At

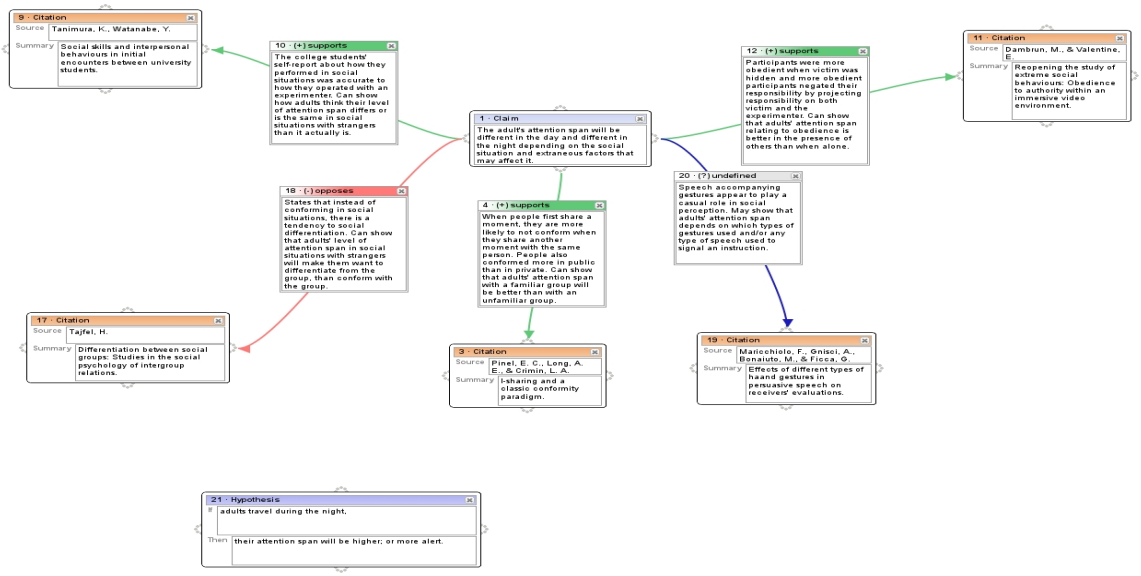
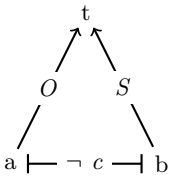


Figure 1: A student-produced LASAD argument diagram representing an introductory argument.

the bottom of the diagram, there is a single isolated *hypothesis* node that contains two text fields, one for a *conditional* or IF field, and the other for *conditional* or THEN field. We expect the induced graph grammars from a set of argument diagrams can be used to evaluate the student thesis work.

Figure 2 shows an *a-priori* rule that was defined as part of that work. This rule is designed to identify a subgraph where a single target node t is connected to two separate citation nodes a and b such that: a is connected to t via an opposing path; b is connected via a supporting path; and there exists no comparison arc between a and b. The rules in that study were implemented using *AGG* an augmented graph grammar library built in Python [12]. AGG matches the graphs using recursive stack-based algorithm. The code first matches the ground nodes at the top-level of the class (t, a, & b). It then tests for the recursive productions O, and S, before finally testing for the negated comparison arc c. This rule does not make use of the full range of potential capacity for Augmented Graph Grammars. However it is illustrative of the type of rules we plan to induce here, rules that generalize beyond basic types and draw on existing production classes but not, at least in the immediate term, use complex textual elements or functional features.

$(ParedWcomp)$



$$\left\{\begin{array}{c} t.Tpye = \text{“}claim\text{”} or \text{“}hypothesis\text{”} \\ a.Type = \text{“}Citation\text{”} \\ b.Type = \text{“}Citation\text{”} \\ c.Type = \text{“}Comparison\text{”} \end{array}\right\}$$

Figure 2: A simple augmented graph grammar rule that detects compared counterarguments. The rule shows a two citation nodes (*a, & b*) that have opposing relationships with a shared claim node (*t*) and do not have a comparison arc (*c*) drawn between them. The arcs *S* and *O* represent recursive supporting and opposing path.

3. GRAMMAR INDUCTION

Graph and relational data has grown increasingly prevalent and graph analysis algorithms have been applied in a wide range of domains from social network analysis [15] to bioinformatics [16]. Most of this work falls into one of two categories of algorithms: frequent subgraph matching, and graph compression.

A number of algorithms have been developed to discover frequent subgraphs. These include the gSpan algorithm developed by Yan and Han [17]; the AGM algorithm developed by Inokuchi et al [18]; and the FSG algorithm developed by Kuramochi and Karypis which is based on the previous Apriori algorithm [19]. They are based upon controlled graph walks coupled with indexing. While these algorithms are effective, particularly on smaller graphs, with low vertex degree they can also overfit simpler graph structures and they do not scale well to larger, denser graph data [20].

The SUBDUE system, originally developed in 1994, takes a greedy-compression approach to graph mining. SUBDUE searches for candidate subgraphs that can best compress the input graphs by replacing a candidate subgraph with a single vertex. Then nodes and arcs are added to the vertices to form new candidate subgraphs. The process is recursive and relies on the Minimum-Description-Length (MDL) principle to evaluate the candidates. SUBDUE has been applied successfully to extract structure from visual programming [5],

web search [21], and analyzing user behaviors in games [22].

While these methods are successful they have practical and theoretical limitations. Both classes of approaches are limited to static graphs composed from a finite alphabet of node and arc types. The frequent subgraph approaches are based upon iterative graph walks and can be computationally expensive and are limited to finding exact matches. They do not generalize beyond the exact graphs matched nor do they allow for recursive typing. SUBDUE, by contrast is a greedy algorithm that finds the single most descriptive grammar and does not allow for weighted matches.

For our present purposes, however, our goal is to identify multiple heirarchical classes of the type shown in Figure 2 that can: generalize beyond exact node and arc types; can draw on recursive rule productions; and can be weighted based upon the graph quality. Moreover our long-term goal with this work is to explore graph rule induction mechanisms that can be expanded to include textual rules and complex constraints. For that reason we have elected to apply evolutionary computation. This is a general-purpose machine learning mechanism that can be tunes to explore a range of possible induction mechanisms.

4. METHODS

4.1 Evolutionary Computation

Evolutionary Computation (EC) is a general class of machine learning and optimization methods that are inspired by the process of Darwinian evolution through natural selection [23] such as Genetic Algorithms [24] or Genetic Programming [25]. EC algorithms begin with a population of randomly generated candidate solutions such as snippets of random code, strings representing a target function, or formal rules. Each of these solutions is ranked by a *fitness function* that is used to evaluate the quality of the individuals. These functions can be defined by absolute measures of success such as a suite of test cases, or by relative measures such as a competition between chess-playing systems.

Once the individuals have been ranked a new generation of individuals is produced through a combination of crossover and mutation operations. *Crossover* operations combine two or more parents to produce one or more candidate children. In Genetic Algorithms where the candidate solutions are represented as strings this can be accomplished by splitting two parents at a given index and then exchanging the substrings to produce novel children. In Genetic Programming the parents exchange blocks of code, functions, or subtrees. *Mutation* operations alter randomly-selected parts of a candidate solution by swapping out one symbol or instruction for another, adding new sub-solutions, or deleting components. This process of ranking and regeneration will iterate until a target performance threshold is reached or a maximum number of generations has passed.

EC methods are highly general algorithms that can be readily adapted to novel domains by selecting an appropriate solution representation and modification operations. Thus, in contrast to more specific methods such as SUBDUE, the EC algorithm allows us to tune the inductive bias of our search and to explore alternative ways of traversing the solution space. Therefore it is well suited to our present needs. This flexibility is costly, however, as EC is far more computationally expensive than more specialized algorithms, and applications of EC can require a great deal of tuning for each use. In the subsections below we will describe the fitness function and the operators that we will use in this work. For this work we will rely on *pyEC* a general purpose evolutionary computation engine that we have developed [26].

4.2 Dataset

Our initial analysis will be based upon a corpus of expert graded student produced argument diagrams and essays previously described in [13, 14]. That dataset was collected as part of a study on students' use of argument diagrams for writing that was conducted at the University of Pittsburgh in 2011. For that study we selected a set of students in an undergraduate-level course on Psychological Research Methods. As part of the course the students were tasked with planning and executing an empirical research study and then drafting a written report. The students were permitted to work individually or in teams. This report was structured as a standard empirical workshop paper. Prior to drafting the report the students were tasked with diagramming the argument that they planned to make using LASAD an online tool for argument diagramming and annotation.

Subsequent to this data collection process the diagrams and essays were graded by an experienced TA using a set of parallel grading rubrics. These rubrics focused on the quality of the arguments in the diagrams and essays and were used to demonstrate that the structure and quality of the diagrams can be used to predict the students' subsequent essay performance. These grades will be used as the weighting metric for the diagrams and will be correlated with performance as part of the fitness function we describe below. After completion of the data collection, grading, and testing phases and accounting for student dropout and incomplete assignments we collected 105 graded diagram-essay pairs 74 of which were authored by teams.

4.3 Solution Representation

For the purposes of our present experiments we will use a restricted solution representation that relies on a subset of the augmented graph grammar formalism exemplified by the rule shown in Figure 2. This will include only element types and recursive productions. In future work we plan to support the induction of more complex rules defined by multiple graph classes, novel productions, and expressions. However for the present study we will focus on the simple case of individual classes coupled with predefined productions.

4.4 Fitness Function

We plan to use the frequency correlation metric previously employed in [13, 14]. In that study the authors assessed the *empirical validity* of a set of *a-priori* diagram rules. The validity of each individual rule was assessed by testing the correlation between the frequency of the class in the existing graph and the graph grade. The strength of that correlation was estimated using Spearman's ρ a non-parametic measure of correlation [27]. In that work the authors demonstrated that the *a-priori* rule frequency was correlated with students' subsequent essay grades and showed that the frequencies could be used to predict students' future performance.

4.5 Mutation

Our mutation operator will draw on the predefined graph ontology to make atomic changes to an existing graph class. The change will be one of the following operations:

Change Node change an existing node's type.

Change Arc Change an existing arc's type or orientation.

Delete Node Delete a node and its associated arcs.

Delete Arc Delete an existing arc.

Add Node Add a novel node with a specified type.

Add Arc Add an arc between existing nodes or add with new nodes.

4.6 Crossover

By design the crossover operation should, like genetic crossover, be conservative. Two very similar parents should produce similar offspring. Crossover operations should therefore preserve good building blocks and sub-solutions or *introns* through random behavior [25]. Arbitrary graph alignment and crossover is a challenging problem that risks causing unsustainable changes on each iteration. We therefore treat graph crossover as a matrix problem.

For each pair of parent classes we will define a pair of diagonal matricies of the type illustrated in Figure 3. The letter indicies on the top and right indicate nodes while the numerical indicies internally indicate arcs, and the $\emptyset$ symbol indicates that no arc is present. The matricies are generated in a canonical order based upon the order in which the nodes were added to the class. Thus on each iteration of the crossover process the corresponding elements will obtain the same index. As a consequence good subsolutions will obtain the same location and will tend to be preserved over time.

Once a set of parent matricies has been generated we then generate two child matricies of the same size as the parents and then randomly select the node and arc members. In the example shown in figures 3 and 4 the parents have nodes $\{A,B,C,D\}$ and $\{E,F,G\}$ while the children have $\{E,B,G,D\}$ and $\{A,F,C\}$. Thus we align the nodes in canonical order and, for each node pair, we flip a coin to decide where they are copied. Excess nodes, in this case D are copied to the larger parent. We then perform a comparable exchange process for the arcs. Each arc or potential arc is defined uniquely in the matricies by its endpoints. We thus align the lists of arcs in a comparable manner and then decide randomly which arc, or empty arc, to copy. As with the excess nodes extra arcs are copied directly into the larger of the two parents.

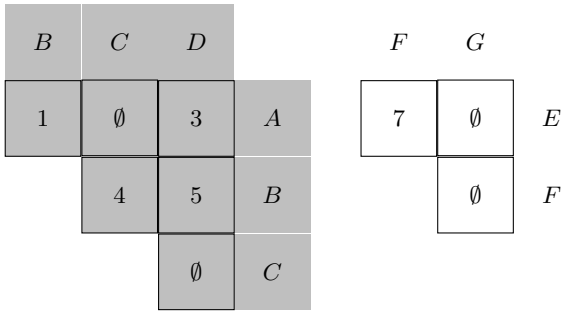


Figure 3: Canonical matricies for crossover parents.

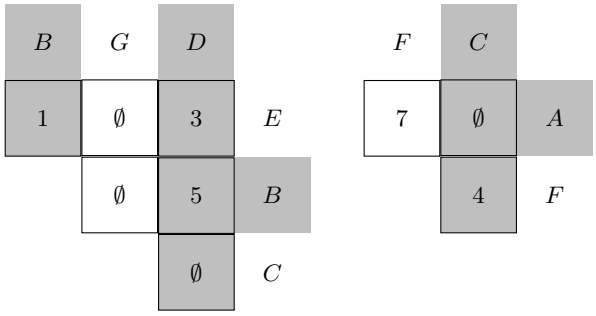


Figure 4: Canonical matricies for crossover children.

5. FUTURE WORK

In this paper we presented a method for the induction of augmented graph grammars through evolutionary computation. We are presently applying this work to the automatic induction of empirically-valid rules for student-produced argument diagrams. This work will serve to extend our prior efforts on the use of augmented graph grammars for student grading and feedback. This work represents an improvement over prior graph grammar induction algorithms which are limited to classical graph grammars and greedy extraction. This work also represents an extension for evolutionary computation by shifting it into a new domain. As part of this work we also plan to explore additional extensions to the standard evolutionary computation algorithm to address problems of over-fitting such as χ^2 reduction.

6. REFERENCES

[1] John L Pfaltz and Azriel Rosenfeld. Web grammars. In *Proceedings of the 1st international joint conference on Artificial intelligence*, pages 609–619. Morgan Kaufmann Publishers Inc., 1969.

[2] John L Pfaltz. Web grammars and picture description. *Computer Graphics and Image Processing*, 1(2):193–220, 1972.

[3] Horst Bunke. Attributed programmed graph grammars and their application to schematic diagram interpretation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 4(6):574–582, 1982.

[4] Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph*. *Data mining and knowledge discovery*, 11(3):243–271, 2005.

[5] Keven Ates, Jacek Kukluk, Lawrence Holder, Diane Cook, and Kang Zhang. Graph grammar induction on structural data for visual programming. In *Tools with Artificial Intelligence, 2006. ICTAI'06. 18th IEEE International Conference on*, pages 232–242. IEEE, 2006.

[6] Francesc Rosselló and Gabriel Valiente. Graph

transformation in molecular biology. In *Formal Methods in Software and Systems Modeling*, pages 116–133. Springer, 2005.

[7] Luc Dehaspe, Hannu Toivonen, and Ross D King. Finding frequent substructures in chemical compounds. In *KDD*, volume 98, page 1998, 1998.

[8] Stefan Kramer, Luc De Raedt, and Christoph Helma. Molecular feature mining in hiv data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 136–143. ACM, 2001.

[9] Wenke Lee and Salvatore J Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM transactions on Information and system security (TiSSEC)*, 3(4):227–261, 2000.

[10] Calvin Ko. Logic induction of valid behavior specifications for intrusion detection. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 142–153. IEEE, 2000.

[11] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[12] Collin F Lynch. Agg: Augmented graph grammars for complex heterogeneous data. In *Workshop on Graph-Based Educational Data Mining*.

[13] Collin F. Lynch and Kevin D. Ashley. Empirically valid rules for ill-defined domains. In John Stamper and Zachary Pardos, editors, *Proceedings of The* 7^{th} *International Conference on Educational Data Mining (EDM 2014)*. International Educational Datamining Society IEDMS, 2014.

[14] Collin F. Lynch, Kevin D. Ashley, and Min Chi. Can diagrams predict essay grades? In Stefan Trausan-Matu, Kristy Elizabeth Boyer, Martha E. Crosby, and Kitty Panourgia, editors, *Intelligent Tutoring Systems*, Lecture Notes in Computer Science, pages 260–265. Springer, 2014.

[15] Sherry E. Marcus, Melanie Moy, and Thayne Coffman. Social network analysis. In Diane J. Cook and Lawrence B. Holder, editors, *Mining Graph Data*, chapter 17, pages 443–468. John Wiley & Sons, 2006.

[16] Chang Hun You, Lawrence B. Holder, and Diane J. Cook. Dynamic graph-based relational learning of temporal patterns in biological networks changing over time. In Hamid R. Arabnia, Mary Qu Yang, and Jack Y. Yang, editors, *BIOCOMP*, pages 984–990. CSREA Press, 2008.

[17] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 721–724. IEEE, 2002.

[18] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Principles of Data Mining and Knowledge Discovery*, pages 13–23. Springer, 2000.

[19] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 313–320. IEEE, 2001.

[20] MICHIHIRO Kuramochi and George Karypis. Finding topological frequent patterns from graph datasets. *Mining Graph Data*, pages 117–158, 2006.

[21] Nitish Manocha, Diane J Cook, and Lawrence B Holder. Cover story: structural web search using a graph-based discovery system. *intelligence*, 12(1):20–29, 2001.

[22] Diane J. Cook, Lawrence B. Holder, and G. Michael Youngblood. Graph-based analysis of human transfer learning using a game testbed. *IEEE Trans. on Knowl. and Data Eng.*, 19:1465–1478, November 2007.

[23] Charles Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray: Albermarle Street: London, United Kingdom, 6 edition, 1872.

[24] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press: Cambridge, Massachusetts, 1999.

[25] Wolfgang Banzhaf. *Genetic programming: an introduction on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers ; Heidelburg : Dpunkt-verlag; San Francisco, California, 1998.

[26] Collin F. Lynch, Kevin D. Ashley, Niels Pinkwart, and Vincent Aleven. Argument graph classification with genetic programming and c4.5. In Ryan Shaun Joazeiro de Baker, Tiffany Barnes, and Joseph E. Beck, editors, *The 1st International Conference on Educational Data Mining, Montreal, Québec, Canada, June 20-21, 2008. Proceedings*, pages 137–146, 2008.

[27] Wikipedia. Spearman's rank correlation coefficient — wikipedia, the free encyclopedia, 2013. [Online; accessed 27-February-2013].