AGG: Augmented Graph Grammars for Complex Heterogeneous Data

Collin F. Lynch
Intelligent Systems Program
and Learning Research & Development Center
Pittsburgh, Pennsylvania, U.S.A.
collinl@cs.pitt.edu

ABSTRACT

The central goal of educational datamining is to derive crucial pedagogical insights from student, course, and tutorial data. Real-world educational datasets are complex and heterogeneous comprising relational structures, social connections, demographic information, and long-term assignments. In this paper I describe *Augmented Graph Grammars* a robust formalism for graph rules that provides a natural structure for evaluating complex heterogeneous graph data. I also describe *AGG* an Augmented Graph Grammar engine written in Python and briefly describe its use.

Keywords

Augmented Graph Grammars, Graph Analysis, Argument Diagrams, Complex Data, Heterogeneous Data

1. INTRODUCTION

The central goal of educational datamining is to draw pedagogical insights from real-world student data, insights which can inform instructors, students, and other researchers. While robust analytical formalisms have been defined for categorical, numerical, and relational data most real-world educational data is complex and heterogeneous combining textual, numerical, and relational features. In large course settings such as a lecture course or MOOC, for example, students may form dynamic working groups and collaborate on complex assignments. They may also be given a flexible set of reading, writing, or problem-solving tasks that they can choose to complete in any order. This process data can be encoded as a graph with nodes representing individual assignments and reading materials and arcs representing group relationships or traversal order. In order to capture important features of this rich graph data and to identify key relationships between teamwork, written text, and performance, it is necessary to apply a rule structure that can capture them naturally.

Individual student assignments can also contain heterogeneous data. Argument diagrams, for example, have been used to teach writing, argumentation, and scientific reasoning [10, 2, 19]. These structures reify real-world arguments as graphs using complex node and arc types to represent argumentative components such as hypothesis statements, citations, and claims. These complex elements can include types, text fields for short notes or free-text assertions, links to external resources, and other data.

A sample student-produced argument diagram drawn from my thesis work at the University of Pittsburgh is shown in Figure 1. This work focused on the use of argument diagrams to support students in developing written scientific reports and in identifying *pedagogically-relevant* diagram structures that can be used to predict students' subsequent performance (see [8]). The diagram contains a central *claim* node representing a research claim. This node has a single text field in which the claim is stated. This is, in turn, connected to a set of *citation* nodes representing related work via a set of *supporting*, *opposing*, and *undefined* arcs colored green, red, and grey, respectively. The citation nodes each contain two text fields, one for the citation information and the other for a summary of the cited work, while the arcs contain a single text field for the *warrant* or explanation of why the relationship holds. At the top of the diagram there is a single disjoint *hypothesis* node which contains two text fields: a *conditional* or IF field, and a *conditional* or THEN field.

This diagram contains a number of pedagogically-relevant issues. Some of them are purely structural such as the disjoint hypothesis node, and the fact that the supporting and opposing arcs are drawn from the claim to the citations and not vice-versa. It also contains more complex semantic issues such as the fact that the text fields on the arcs contain summary information for the cites not explanations of the relationship, and the fact that the opposing citations, citations that disagree about the central claim node have not been distinguished from one-another via a comparison arc. Problems such as these can be detected via complex rules, and I have previously shown that the presence of such problems are predictive of students' subsequent performance [8, 10, 9]. This detection and remediation, however requires the development of rules that can incorporate complex structural and textual information.

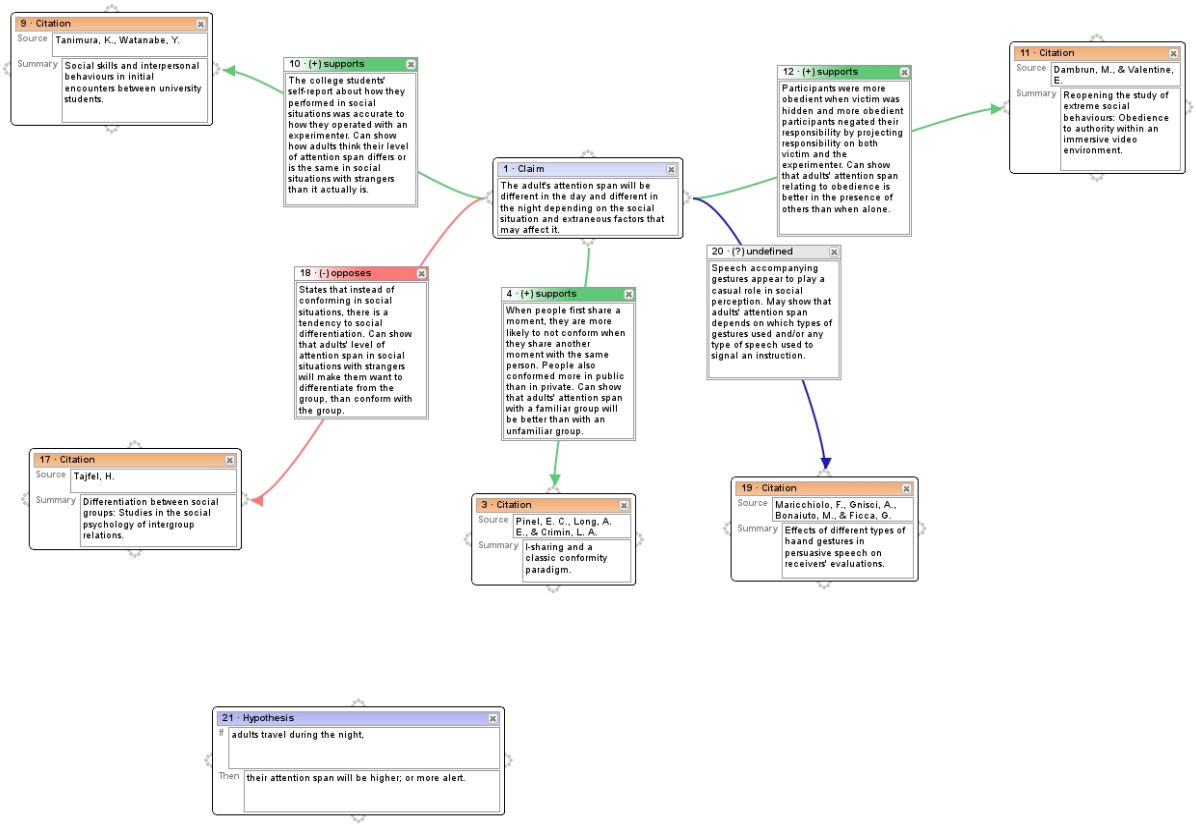

Figure 1: A segment of a student-produced LASAD diagram representing an introductory argument. It contains a central *claim* node surrounded by *citation* nodes. The isolated node is a *hypothesis* that has not been integrated into the argument.

Automatic graph analysis is central to a number of research domains including strategy transfer in games [4], automatic recommendations [1], cheminformatics [12], and social network detection [11]. Graph analysis algorithms have been used to define educational communities [15, 16, 5]) and to automatically grade existing datasets [8, 10, 9]. Graphical structures have also been used in tutoring contexts to represent student work via argument diagrams of the type shown above (see [14, 7] or to provide connection representations [19] for student guidance.

My focus in the present work is on the development of graph *rules* that is logical graph patterns that match arbitrary graph structures based upon content and structure information. While arbitrary graph matching is NP-Hard (see [18]) it is of practical importance, particularly in relational domains such as argument diagrams or student groups where our goal is to identify complex structures that may be evidence of deeper pedagogical issues. To that end, I will introduce *Augmented Graph Grammars* a robust rule formalism for complex graph rules and will describe *AGG* and augmented graph grammar engine for educational datamining. Both were developed as part of my thesis work at the University of Pittsburgh.

2. AUGMENTED GRAPH GRAMMARS

Graph Grammars, as described by Rekers and Schürr, are formal grammars whose atomic components are graphs or graph elements, and whose productions transpose one graph to another [17]. More formally, they define graph-grammars and productions as:

Definition 3.6 A graph grammar GG is a tuple $(A; P)$, with A a nonempty initial graph (the axiom), and P a set of graph grammar productions. To simplify forthcoming definitions, the initial graph A will be treated as a special case of a production with an empty left-hand side. The set of all potential production instances of GG is abbreviated with $PI(GG)$.

Definition 3.2 A (graph grammar) production p := (L; R) is a tuple of graphs over the same alphabets of vertex and edge labels LV and LE. Its left-hand side lhs(p) := L and its right-hand side rhs(p) := R may have a common (context) subgraph K if the following restrictions are fulfilled:

- $\forall e \in E(K) \Rightarrow s(e) \in V(K) \wedge t(e) \in E(K)$ with $E(K) := E(L) \cap E(R)$ and $V(K) := V(L) \cap V(R)$ i.e. sources and targets of common edges are common verticies of L and R, too.

- $\forall x \in L \cap R \Rightarrow l_L(x) = l_R(x)$ i.e. common elements of L and R do not differ with respect to their labels in L and R.

Thus graph grammars are systems of production rules analogous to context-sensitive string grammars (see [18]). For reasons of efficiency Rekers and Schürr restrict their focus to *layered graph-grammars* where all productions must be *expansive* with the left-hand-side being a subgraph of the right. Classical graph grammars, like string grammars, assume a fixed alphabet of simple statically-typed node and arcs and can be used both to generate matching graphs programmatically or to parse matching graphs via mapping and decomposition. My focus in the present work is on graph matching which occurs via iterative mapping.

Let $G_i =< \{n_o, \ldots\}, \{e(n_p, n_q), \ldots\} >$ and $G_j =< \{m_o, \ldots\}, \{e(m_k, m_l), \ldots\} >$ be graphs and let $M = \{< n_a, M - b > \ldots\}$ me a *mapping* from G_i to G_j that links nodes of the two. In the context of a mapping, G_i and G_k are called the *source* and *target* graphs respectively. A mapping M_{G_i,G_j} from G_i to G_j is *valid* if and only if the following holds:

$$\forall n_x \in G_i : \exists < n_x, m_y > \in M_{G_i,G_j}$$

$$\neg\exists\{< n_x, m_y >, < n_r, m_k >\} \subseteq M_{G_i,G_j} : (x = r) \vee (y = k)$$

$$\forall e(n_x, n_y) \in G_i : \{< n_x, m_y >, < n_r, m_k >\} \subseteq M_{G_i,G_j} : \exists e(m_y, m_k) \in G_j$$

For the remainder of this paper all elements in a source graph will be labeled alphabetically (e.g. a, Q) while elements in the target graphs will be referenced numerically (e.g. $1, 2, e(2,3), \overrightarrow{e(4,5)}$).

Augmented Graph Grammars are a richer formalism for graph rules that treat nodes and arcs as complex components with optional sub-fields including flexible text elements or other types. Augmented graph grammars have been previously described by Pinkwart et al. in [13]. There the authors focused on the use of augmented graph grammars for tutoring. An Augmented Graph Grammar is defined by: a graph ontology that specifies the complex graph elements and functions available; a set of graph classes that define matching graphs; and optional graph productions and expressions that provide for recursive class mapping and logical scoping. I will describe each of these components briefly below. For a more detailed description see [8].

2.1 Graph Ontology

In a simple graph grammar of the type used by Rekers and Schürr the set of possible node and arc types ($\sum$) is fixed with the elements being atomic, static, and unique. In order to process complex structures such as the argument diagram shown in Figure 1, a more complex structure is required. Thus augmented graph grammar ontologies are defined by a set of element types $O = \{N_0, \ldots N_m, E_0, \ldots, E_p\}$ such that each element has a unique list of fields and field types as well as applicable functions over those fields. The ontology must also specify appropriate relationships between the fields and operations that can be used on them.

While showing a complete ontology is beyond the scope of this paper an illustrative example can be found in Figure 2. This illustrates the field definitions for the citation and hypothesis nodes shown above. Both node types contain two sub-fields of type String. For each of these fields an additional function is defined '*.Words' which returns a set of all the words found in the field.

```
{
 Nodes:{
   Citation:{
     Cite(String)
     Cite.Words(StringSet)
     Summary(String)
     Summary.Words(StringSet)
   }

   Hypothesis: {
     If(String)
     If.Words(StringSet)
     Then(String)
     Then.Words(StringSet)
   }
 }
Arcs:{
  Comparison: {
   ...

Types: { String, StringSet }
...
```

Figure 2: An illustrative subset of a sample graph ontology for scientific argument diagrams.

2.2 Graph Classes

The core component of an augmented graph grammar is the graph *class*. A class C_i is defined by a 2-tuple $< S_i, O_i >$ where S_i is a graph *schema* and O_i is a set of *constraints*. A class defines a space of possible graphs which satisfy both the schema and the constraints. Classes are not required to be unique nor are the set of matching graphs for a given pair of classes required to be disjoint. A sample named class $R07a$ is shown in 3. This class is designed to detect instances of *Related Uncompared Opposition* in scientific argument diagrams. That is subgraphs where there exists a pair of citation nodes a, and b that disagree about a shared target node t, are not connected via a comparison arc c, and which share some relevant textual content. As I noted above, this type of structure can be found in Figure 1.

2.2.1 Graph Schema

A *Schema* is a graph structure that defines a space of possible graphs topologically. Schema are defined by a set of *ground nodes* (e.g. t, a, & b in Figure 3) which must match a single node in a target graph, a set of *ground arcs* that must likewise match a single arc in the target graph (e.g. c), and an optional set of *variable arcs* which must match a nonempty subgraph defined by a graph production. By convention, ground elements are denoted via lower-case names while variable elements are denoted by capitalized names.

In addition to the ground and variable distinctions arcs within a schema may be one of four types: *directed* (e.g. O, &

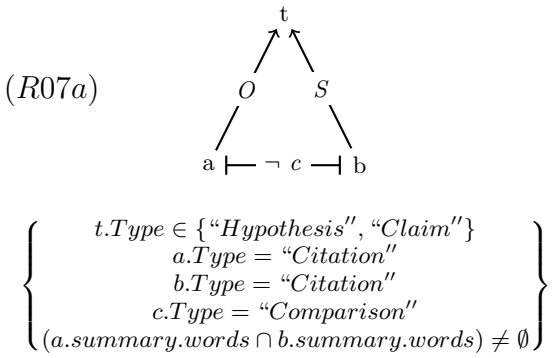

Figure 3: *Related Uncompared Opposition* A simple augmented graph grammar rule that detects related but uncompared counterarguments. The rule shows a two citation nodes (*a*, & *b*) that have opposing relationships with a shared hypothesis or claim node (*t*) and do not have a comparison arc (*c*) drawn between them. The arcs *S* and *O* represent recursive supporting and opposing paths.

S), of *unknown direction*, *undirected* (e.g. *c*), and *undefined*. Directed arcs will only match directed arcs in the base graph oriented in the same direction. Thus, given a base graph containing an arc $e(1,2)$ and a schema with a directed arc $\overrightarrow{e(n,m)}$ the schema will only match cases where $\{<n,1>,<m,2>\} \subseteq M$. Unknown direction schema arcs may match a directed arc oriented in any order but will *not* match an undirected arc (e.g. $e(2,3)$). Undirected arcs (e.g. $\neg c$) will not match a directed arc. And, undefined arcs may match a directed or undirected arc in any order.

As the example shows arcs may be also be negated (e.g. $\neg c$) in which case the schema matches a graph if and only if no match can be found for the negated arc. Thus the schema shown will only match ground graphs with no arc between the elements assigned to *a* and *b*. More complicated cases of negation may be formed using graph expressions which are defined below.

The elements of a Schema must also be *non-repeating* that is, no two elements in a schema may be matched to the same element in the target graph. Thus each element in a schema must match at least one unique node or arc with variable elements possibly accounting for more than one element.

2.2.2 Constraints

Constraints represent individual bounds or limits on the ground elements of a schema. Constraints are specified using a set-theory syntax (e.g. $t.Type \in \{\text{"}Hypothesis\text{"}, \text{"}Claim\text{"}\}$) and may draw on any of the node or arc features, subfields, or functions specified in the ontology. *Unary Constraints* apply to a single element (e.g. $a.Type = \text{"}Citation\text{"}$). *Binary Constraints* (e.g. $(a.summary.words \cap b.summary.words) \neq \emptyset$) specify a relationship between two distinct ground elements.

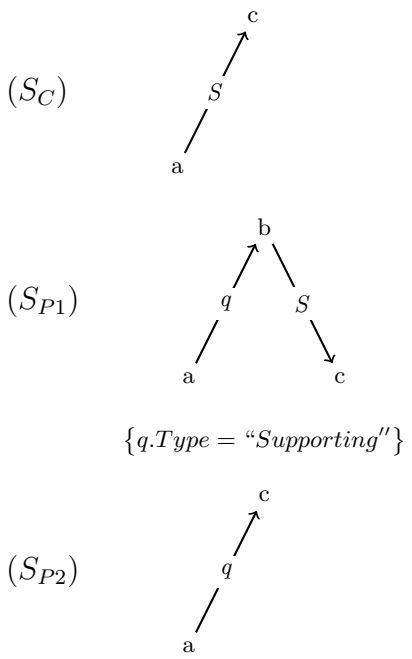

Figure 4: A simple recursive rule production for S that defines a *supporting path*.

2.3 Graph Productions

A graph production $C_l \Rightarrow C_{r1}|C_{r2}...$ is a context-sensitive production rule that maps from a graph class containing a single *production variable* to one or more alternate expansions. Graph productions are used to match layered subgraphs to the variable arcs. A simple recursive production rule for the variable element $\overrightarrow{S(b,t)}$ is shown in Figure 4.

The rule is defined by the *context class* S_C, and the two *production classes* S_{P1} and S_{P2}. The context class is used as a key for the production application. It must contain exactly one variable arc, the *production variable*, and no constraints. The ground nodes *a* and *c* are *context nodes* and are used to ground the production for mapping. They must be present in all of the production rules. All production rules must be *expansive* with each of the production classes containing at least one ground element not present in the context class. Recursive productions are thus handled by iteratively grounding the mapping with additional context and, as per the *non-repeating* requirement, these rules must consume additional elements of the graph. Production rules are thus mapped in a layered fashion like the grammars defined by Rekers and Schürr.

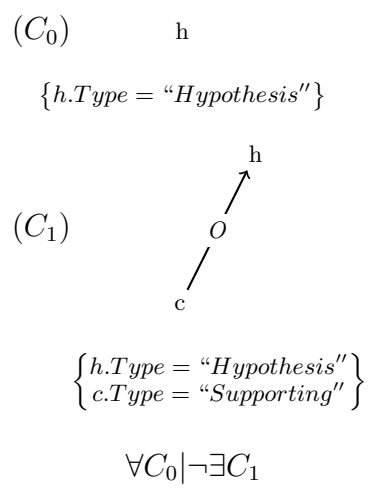

Figure 5: A simple Graph expression that tests for unopposed hypotheses.

2.4 Graph Expressions

Graph expressions are logical rules of the form:

$$S_0C_0 \quad | \quad S_1C_1 \quad | \quad \dots \quad | \quad S_mC_m$$

where each C_i is a graph class and each S_i is a logical quantifier from the set: $\{\forall, \neg\forall, \exists, \neg\exists\}$. The expressions allow for existential and universal scoping and arbitrary negation of graph classes. The expressions represent chained logical structures with each '|' being read as "...such that ...". A sample graph expression is shown in Figure 5. This sample expression asserts that for all hypothesis nodes in the target graph there exist no citation nodes that oppose the target hypothesis. Thus it is a universal claim about a negated existential item. As this example illustrates graph expressions allow for more complex negation structures than are supported by the graph schema.

Graph expressions must be expansive or *right-grounded* such that the following constraints hold:

$$\forall C_{m \leq i > 0} \in E : C_{i-1} \subseteq_g C_i$$

$$S_m \in \{\exists, \neg\exists\}$$

That is, the schema component of class C_i must be a subgraph of all classes class C_{i+n}. This also holds true for the constraints with all constraints present in class C_i being present in classes C_{i+n}. And the rightmost class in the expression must also be an existential ($\exists$) test with optional negation.

3. AGG

AGG is a general-purpose augmented graph grammar engine that implements recursive graph matching. The system was developed in Python to support analysis of the student-produced argument diagrams described above. As such it is flexible, functions across platforms, and supports complex graph ontologies and user-defined functions. The system was designed in a modular fashion and can be linked with third-party libraries such as the NLTK [6].

At present the system uses a straightforward depth-first stack matching algorithm. Given a graph and a set of named *rules*, defined by a single graph class or expression, the system will first match all ground nodes and arcs in the leftmost *target* class. Once each ground element has been matched then the system will recursively match all variable elements in the target. If at any point the system cannot continue to match elements it will pop up the stack and repeat. Rule matching is governed by the aforementioned restrictions of expansiveness and non-repetition. If a rule is defined by a graph expression then each class match will set the context for subsequent rightmost matches. Rules defined by a single class are complete once a single match is found. The system is designed to find matches serially and can be called iteratively to extract all matching items.

In addition to basic graph grammars the AGG toolkit has the capacity to define named rules. These are named graph expressions or individual classes that will be recorded if they match. In my thesis work, I applied the AGG engine to develop a set of 42 such rules the scientific argument diagrams. These ranged in complexity from graph classes defined by a single node to more complex recursive expressions that sought to identify disjoint subgraphs and unsupported hypotheses. The example rules and expressions shown in figures 3 - 5 were adapted from this set. The rules were used for offline processing of the graphs and for prediction of student grades [10, 9].

As part of the analysis process the rules were evaluated on a set of 526 diagrams containing between 0 and 41 nodes each. While exact efficiency data was not retained the performance of the rules varied widely depending upon their construction. General recursive rules such as a test for disjoint subgraphs performed quite inefficiently while smaller chained expressions were able to evaluate in a matter of seconds on a quad-core system.

4. APPLICATIONS & FUTURE WORK

The focus of this paper was on introducing Augmented Graph Grammars and the AGG engine. The formalism provides for a natural and robust representation of complex graph rules for heterogeneous datasets. In prior work at the University of Pittsburgh I applied Augmented Graph Grammars to the detection of pedagogically relevant structures like *Related Uncompared Opposition* (see Figure 3) in argument diagrams of the type shown in Figure 1. The focus of that study was on testing whether student-produced argument diagrams are diagnostic of their ability to produce written argumentative essays. The study was conducted in a course on Psychological Research Methods at the University of Pittsburgh.

The graph features examined in that study included *chained counterarguments* which feature chains of oppositional information, and *ungrounded hypotheses* which are unrelated to cited works, and so on. The study is described in detail in [8], and a discussion of the empirical validity of the individual rules can be found in [9]. The rules were also used as the basis of predictive models for student grades described in [10]. The Augmented Graph Grammars were ideally-suited for this task as they allowed me to define clear and robust rules that incorporated the structural information in the graph, textual information within the nodes and arcs, and the static

element types. It was also possible to clearly present these rules to domain experts for evaluation.

While the AGG system is robust more work remains to be done to make it widely available, and several open problems remain for future development. As noted above, arbitrary graph parsing is NP-Hard. Consequently, many rule classes are extremely inefficient. Despite this limitation, however, real efficiency gains may be made via parallelization and memoization. I am presently researching possible improvements to the system and plan to test them with additional datasets.

Acknowledgments

This work was supported by National Science Foundation Award No. 1122504, "DIP: Teaching Writing and Argumentation with AI-Supported Diagramming and Peer Review," Kevin D. Ashley PI with Chris Schunn and Diane Litman, co-PIs.

5. REFERENCES

[1] Euijin Choo, Ting Yu, Min Chi, and Yan Lindsay Sun. Revealing implicit communities to incorporate into recommender systems. In *Proceedings of the* 15^{th} *ACM Conference on Economics and Computation*, Palo Alto, CA, 2014. Association For Computing Machinery. (in press).

[2] Evi Chryssafidou and Mike Sharples. Computer-supported planning of essay argument structure. In *Proceedings of the 5th International Conference of Argumentation*, June 2002.

[3] Diane J. Cook and Lawrence B. Holder, editors. *Mining Graph Data*. John Wiley & Sons, 2006.

[4] 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.

[5] Rosta Farzan and Peter Brusilovsky. Annotated: A social navigation and annotation service for web-based educational resources. *Journal of the New Review of Hypermedia and Multimedia (NRHM)*, 2008.

[6] Dan Garrette, Peter Ljunglöf, Joel Nothman, Mikhail Korobov, Morten Minde Neergaard, and Steven Bird. The natural language toolkit for python (NLTK), 2014. [Online; accessed 04-29-2014].

[7] Frank Loll and Niels Pinkwart. Lasad: Flexible representations for computer-based collaborative argumentation. *Int. J. Hum.-Comput. Stud.*, 71(1):91–109, 2013.

[8] Collin F. Lynch. The Diagnosticity of Argument Diagrams, 2014. (defended January 30th 2014).

[9] 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. (In Press).

[10] Collin F. Lynch, Kevin D. Ashley, and Min Chi. Can diagrams predict essays? In Stefan Trausan-Matu, Kristy Elizabeth Boyer, Martha E. Crosby, and Kitty Panourgia, editors, *Intelligent Tutoring Systems*, volume 8474 of *Lecture Notes in Computer Science*, pages 260–265. Springer, 2014.

[11] Sherry E. Marcus, Melanie Moy, and Thayne Coffman. Social network analysis. In Cook and Holder [3], chapter 17, pages 443–468.

[12] Takashi Okada. Mining from chemical graphs. In Cook and Holder [3], chapter 14, pages 347–379.

[13] Niels Pinkwart, Kevin D. Ashley, Vincent Aleven, and Collin F. Lynch. Graph grammars: An its technology for diagram representations. In David Wilson and H. Chad Lane, editors, *FLAIRS Conference*, pages 433–438. AAAI Press, 2008.

[14] Niels Pinkwart, Kevin D. Ashley, Collin F. Lynch, and Vincent Aleven. Evaluating an intelligent tutoring system for making legal arguments with hypotheticals. *International Journal of Artificial Intelligence in Education*, 19(4):401–424, 2009.

[15] Eve Powell and Tiffany Adviser-Barnes. A framework for the design and analysis of socially pervasive games. 2012.

[16] Eve M Powell, Samantha Finkelstein, Andrew Hicks, Thomas Phifer, Sandhya Charugulla, Christie Thornton, Tiffany Barnes, and Teresa Dahlberg. Snag: social networking games to facilitate interaction. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pages 4249–4254. ACM, 2010.

[17] J. Rekers and Andy Schürr. Defining and parsing visual languages with layered graph grammars. *J. Vis. Lang. Comput.*, 8(1):27–55, 1997.

[18] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, San Francisco, 1997.

[19] Daniel D. Suthers. Empirical studies of the value of conceptually explicit notations in collaborative learning. In Alexandra Okada, Simon Buckingham Shum, and Tony Sherborne, editors, *Knowledge Cartography*, pages 1–23. Springer Verlag, 2008.