ON THE DISCOVERY OF MATHEMATICAL THEOREMS

Susan L. Epstein

Department of Computer Science
Hunter College of the City University of New York
695 Park Avenue, New York, NY 10021

ABSTRACT

The Graph Theorist, GT, is a system which performs mathematical research in graph theory. This paper focuses upon GT's ability to conjecture and prove mathematical theorems from the definitions in its input knowledge base. Each class of graphs is defined in an algebraic notation with a semantic interpretation that is a stylized algorithm to generate the class correctly and completely. From a knowledge base of such concept definitions, GT is able to conjecture and prove such theorems as "The set of acyclic, connected graphs is precisely the set of trees" and "There is no odd-regular graph on an odd number of vertices." Conjecture and proof are driven both by examples (specific graphs) and by definitional form (algorithms).

1. Introduction

The Graph Theorist, GT, is a knowledge-intensive, domain-specific learning system which uses algorithmic class descriptions to discover and prove relations among mathematical concepts, i.e., theorems, in graph theory. Although it primarily exemplifies theory-driven discovery, where search heuristics postulate and test the validity of conjectures, it also infers explanations for factual input about key graphs as a data-driven system would.

Mathematical discovery includes the creation of new mathematical concepts, the conjecture of relations among concepts and the proof or disproof of such conjectures. Thus far, AI discovery in mathematics has focused attention primarily on the deductive proof of mathematical theorems using a predicate calculus representation. The notable exception has been Lenat's AM[Len76]. AM generated mathematical objects and observed statistical regularities in their classification. When the experimental evidence warranted it, AM would conjecture relations among classes. Discovery in AM was driven not only by empirical evidence but also by 243 heuristic rules which originated the generation of examples and new concepts, and evaluated the results. Later it was generally recognized[Len84,Rit] that the path of discovery was pronouncedly affected by the programming language itself.

In this sense, AM may be said to model some intuition on the part of the research mathematician as to a representation language (LISP) well-suited for exploration in a particular direction (number theory). AM's results indicate that, if knowledge about a domain can be semantically encoded into the class definition, then it can be harnessed to drive mathematical discovery. GT encodes the semantics of graph theory into class definitions in a more transparent and flexible fashion, one which supports both inductive and deductive reasoning. These semantics subsequently motivate both conjecture and proof.

2. Concept Description in GT

The material in this section describes the application of a theoretical formulation detailed rigorously in [Eps83] and summarized in [Eps87]. The treatment here is informal and describes only selected, implemented segments of the theory. For example, GT currently only supports undirected, unlabelled graphs, but coding provisions have been made for directed and labelled graphs, and the theoretical framework supports them. Let V be an arbitrary, finite set of elements (*vertices*) and let E be any subset (*edges*) of the Cartesian product V × V. Then the ordered pair G = <V,E> is said to be a *graph*. Let U be the universe of all graphs. Then any subset P of U is said to designate a *graph property p* and, for G in P, G is said to *have property p*. Any algorithmic definition of the graph property p must specify precisely the set P. In particular, if an algorithm claims to generate P, that algorithm must be both *correct* (i.e., every generated graph must be in P) and *complete* (i.e., for each graph G in P there must be a finite sequence of steps executed by the algorithm with final output G).

In GT, a *concept* is a frame representing a graph property and knowledge associated with it, as in [Michen]. A slightly-edited example of an input GT frame for the concept ACYCLIC appears on the left in Figure 1. The slots of the frame include a list of examples, knowledge about hierarchical relations with other concepts, historical information on the ways the concept has been manipulated, and a description of the origin of the property. (Entries of "nil" for relations are statements of partial knowledge, to be read as "none discovered yet.") The frame also includes a definition of the graph property in a specific, three-part formulation.

In GT, a *definition* of a graph property is an ordered triple $<f,S,\sigma>$. S is the *seed set*, a set of one or more minimal graphs (*seeds*), each of which has the property in question. (Typically the seed set is finite and GT lists its elements.) The seed set in the example of Figure 1 for ACYCLIC contains only K_1, the complete loop-free graph on one vertex. The *operator* f in the definition describes how any graph with the given property may be transformed to construct another graph with the same property. An operator in GT is built from a set of four primitive operators: add the vertex x (A_x), add the edge between x and y (A_{xy}), delete the vertex x (D_x), and delete the edge between x and y (D_{xy}). These primitives may be concatenated into *terms* (such as "$A_{yz}A_z$") to denote sequential operation from right to left. Terms may be summed (as in "$A_x + A_{yz}A_z$") to represent alternative actions. Thus the operator $A_x + A_{yz}A_z$ for ACYCLIC is read "either add a vertex x or else add a vertex z and then an edge from y to z." The *selector* σ in the definition describes the restrictions for binding the variables appearing in the operator f to the vertices and edges in a graph. In GT, there are currently four valid kinds of selector descriptions for a vertex: whether or not it is in the graph, its distinctness from another specific vertex symbol, its degree (number of neighbors) and whether or not its degree is the maximum among the degrees of all the vertices in the graph. The current valid edge selector descriptions in GT are of two kinds: whether or not the edge is in the graph, and whether its endpoints are distinct.

Selector descriptions may be empty, i.e., need not constrain binding at all. In the example of Figure 1, the selector for ACYCLIC is read "where y is in the vertex set, and x and z are not in the vertex set."

The semantic interpretation of such a three-part definition for a graph property p is a single, uniform algorithm called a *p-generator*. A p-generator capitalizes on the underlying commonality of its class, the view of the set P as one or more prototypes (seeds) which can be methodically transformed (under f and σ) to produce exactly those graphs in the class. The p-generator may be thought of as an automaton which is started by the input of any graph in seed set S. ACYCLIC, for example, would require K_1. The p-generator then iterates an undetermined number of times. On each iteration the selector σ chooses vertices and/or edges with respect to the current graph G, and then the operator modifies G, using those choices, to produce a new G. ACYCLIC, on each iteration, either adds a new vertex x to the graph, or adds a new vertex z and an edge from an old vertex y to z.

Thus the algorithm for generating the class P of graphs is

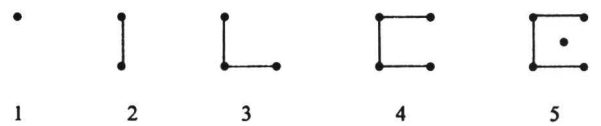
Accept $G \in S$
Output G
Until σ fails do
  $G \leftarrow f\sigma(G)$
  Output G
Halt

Under all possible initial choices from S and all possible iterations of f subject to σ, the output of this algorithm is precisely P, that is, if the superscript i denotes "iterate i times,"

$$P = \bigcup_{i=0}^{\infty} (f\sigma)^i(S)$$

The graphs in Figure 2 illustrate several possible iterations of the definition of ACYCLIC; each pictured graph is output by the algorithm and is acyclic. The definition generates the infinite class of acyclic graphs; it will never halt because bindings for the variables in σ can be found on each iteration.

The content of the following three general texts is taken as *graph theory*: [Ore], a classical development in elegant mathematical fashion; [Har], a broad overview of topics presented as definitions and theorems; and [Bon], an algorithmic approach. What evidence is there that p-generators exist for every P in U, or at least for every interesting P in graph theory? At this writing, more than 40 properties of varying difficulty have been selected from the three benchmark texts and described correctly and completely as p-generators[Eps83]. The use of p-generators as property definitions entails several kinds of non-determinism. Any graph in the seed set is an acceptable input; any binding satisfying σ is valid; any term in f suffices for an iteration. In addition, many different sequences of iterations will construct isomorphic graphs. This ostensible indefiniteness and redundancy is tolerated because the property definitions preserve detail in a concise and flexible format.



GT Iterations of ACYCLIC

FIGURE 2

3. Relations between Concepts in GT

Inductive inference from examples does not preserve truth, only falsity[Mic83]. Although research mathematicians devote much time to example generation, and infer conjectures about relations among ideas based on these examples, inductive inference is only a tool. Rarely is a conjecture considered a result worthy of publication, and then only when extensive attempts at proof and disproof have failed. Mathematicians prefer to explore in the context of certainty; for them a conjecture should be proved or disproved relatively soon after it arises. GT, therefore, constructs both conjectures and proofs.

Graph theory, as it appears in the three benchmark texts cited above, is primarily about graph properties and the relations among them. Conjectures and theorems in graph theory frequently take one of the following forms:

SLOT	INITIAL FORMULATION	AFTER EXECUTION
Property-name:	ACYCLIC	ACYCLIC
Number-of-seeds:	1	1
Seed-set:	$\{K_1\}$	$\{K_1\}$
Function:	$A_x + A_{yz}A_z$	$A_x + A_{yz}A_z$
Sigma:	$y \in V$, $x,z \notin V$	$y \in V$, $x,z \notin V$
Origin:	input	input
Examples:	$\{K_1\}$	{ACYCLIC-3, ACYCLIC-2, K_1}
Extremal-cases:	$\{K_1\}$	$\{K_1\}$
Delta-pairs:	((1 0)(1 1))	((1 0)(1 1))
Subsumes:	nil	(ACYCLIC-MERGED-WITH-CONNECTED CHAIN TREE)
Not-shown-to-subsume:	nil	(CONNECTED EQUIV-CONNECTED)
Subsumed-by:	nil	(IS-A-GRAPH)
Not-shown-subsumed-by:	nil	(ACYCLIC-MERGED-WITH-CONNECTED CONNECTED TREE EQUIV-CONNECTED CHAIN)
Merger-created-with:	nil	(CONNECTED)
Merger-explored-with:	nil	(CHAIN TREE)
Is-equivalent-to:	nil	nil
Not-shown-equivalent-to:	nil	(CHAIN TREE CONNECTED EQUIV-CONNECTED IS-A-GRAPH)

GT Representation of ACYCLIC

FIGURE 1

- TYPE 1: If a graph has property p, then it has property q.
- TYPE 2: A graph has property p if and only if it has property q.
- TYPE 3: If a graph has property p and property q, then it has property r.
- TYPE 4: It is not possible for a graph to have both property p and property q.

GT has two fundamental procedures for manipulating graph properties to prove such theorems. The first procedure tests for subsumption. Property p for class P *subsumes* property q for class Q if and only if Q is a subset of P, i.e., every graph with property q also has property p, so that q is a special case of p. Theorem type 1 is a statement that q subsumes p. Theorem type 2 is a statement that p and q are equivalent, i.e., that p subsumes q and q subsumes p. The second procedure constructs mergers. The *merger* of a property p for class P with a property q for class Q results in a new property representing P∩Q, the set of graphs with both properties. Theorem type 3 is a statement that property r subsumes the merger of p and q. Theorem type 4 is a statement that the merger of p and q is empty, i.e., that no graph can have both properties simultaneously.

3.1 Proof of Subsumption

At this writing, GT has only one method of proving subsumption. Given property $p_1 = \langle f_1,S_1,\sigma_1\rangle$, property $p_2 = \langle f_2,S_2,\sigma_2\rangle$, and a conjecture that p_1 subsumes p_2, GT attempts to show that:

- f_2 is subsumed by f_1, i.e., f_2 is a special case of f_1.
- Every graph in S_2 has property p_1.
- σ_2 is subsumed by σ_1, i.e., σ_2 is more restrictive than σ_1.

(Extended definitions for operator and selector subsumption appear in [Eps83].) Because there are usually only a few known seeds, GT checks the list of examples for p_1 against S_2. If any graph G in S_2 is not known to have p_1, GT generates a limited number of new examples of p_1 and searches for G there. Because seed graphs are extremal cases, and because a natural metric exists on most GT definitions, the search is readily controlled and usually successful. (Alternative techniques exist for infinite seed sets and certain other situations.) Matching for the subsumption testing of the operators and selectors is done by a recursive backtracking algorithm which generates a restricted set of candidates.

The following example illustrates the subsumption procedure. For

$$\text{ACYCLIC} = \langle A_x + A_{yz}A_z, \{K_1\}, [x,z \notin V, y \in V]\rangle$$

and

$$\text{TREE} = \langle A_{pq}A_q, \{K_1\}, [p \in V, q \notin V]\rangle$$

and the conjecture "ACYCLIC subsumes TREE," GT must show that K_1, the seed for TREE, is an acyclic graph and also that, under some matching, the ACYCLIC operator "covers" the TREE operator while satisfying the TREE selection constraints. First, K_1 is on the list of acyclic graphs because it is the seed for ACYCLIC. Second, the matcher notes that every term in the TREE operator $A_{pq}A_q$ (there is only one in this example) is covered by some term, namely $A_{yz}A_z$, in the ACYCLIC operator. Finally, the matcher observes that under the matching of p with y and q with z, the selector constraints (that p is in V and q is not) are enforced. Thus GT proves that ACYCLIC subsumes TREE or, more formally, "Every tree is an acyclic graph."

3.2 Proofs Involving Mergers

GT currently has four algorithms for merger. Given property $p_1 = \langle f_1,S_1,\sigma_1\rangle$ and property $p_2 = \langle f_2,S_2,\sigma_2\rangle$, GT attempts to construct the merger $p = \langle f,S,\sigma\rangle$ of p_1 and p_2. The first three algorithms are fairly straightforward:

- If p_1 subsumes p_2, the merger is simply p_2.
- When f_1 subsumes f_2 and every seed in S_2 has property p_1, the merger is $\langle f_2,S_2,\sigma\rangle$, where σ is σ_1 and σ_2, eliminating any references to variables not in f_2.
- When f_1 subsumes f_2, σ_1 subsumes σ_2, and S is non-empty, the merger is $\langle f_2,S,\sigma_2\rangle$, where

$$S = \{G \mid G \in S_2 \cap P_1\} \cup \{G \mid G \in S_1 \cap P_2\}.$$

The fourth algorithm addresses the more interesting cases which do not fit these categories. Here GT examines how the number of vertices and the number of edges change as the p-generator iterates. GT uses this information in heuristic attempts to create a hybrid generator which satisfies both definitions. (A more detailed description of these merger techniques, with examples, appears in [Eps].)

Some of the most interesting of GT's proofs are merger failures. Consider, for example, GT's discovery that a graph which is odd-regular (every vertex of degree d, and d is odd) cannot have an odd number of vertices. When no common seed is evident, GT generates some examples to expand its list of graphs with an odd number of vertices, seeking one which is odd-regular. When this effort fails, GT considers the possibility that there is no common seed and examines the changes to m and n wrought by the operators. GT recognizes that ODD-NUMBER-OF-VERTICES begins with one vertex and adds two vertices at a time, so that n is always odd, but that ODD-REGULAR begins with an even number of vertices (the seed is K_2) and adds an even number of vertices at a time, so that n is always even. This disparity is the reason GT gives in its proof: there can never be a seed for the merger, and thus the property has no example, i.e., is impossible.

4. Conjecture

A mathematician presented with non-empty classes P and Q from a universe U is trained to explore potential relations between the classes by examining whether or not each of P∩Q, P - Q and Q - P is empty. GT models this strategy with conjectures about subsumption and merger. The standard mathematical questions, and their GT equivalents are

- Is P a subset of Q? GT explores this by a conjecture that q subsumes p.
- Is P a superset of Q? GT explores this by a conjecture that p subsumes q.
- Is P equal (equivalent) to Q? GT explores this by two conjectures, that p subsumes q and that q subsumes p.
- Are P and Q disjoint (mutually exclusive)? GT explores this by a suggestion to merge p and q.

Thus the theorems that GT conjectures are statements about set-theoretic relations between classes of graphs. Given the four theorem types in Section 3 and a knowledge base of k properties, there are potentially 2k(k+1) projects (proposals for exploration) on the first pass, i.e., before newly-created properties participate in project formulation. How does GT limit search through such a space? The human mathematician has two primary sources of evidence on which to base project formulation: examples and definitions. GT is capable of reasoning both from p-generator definitions and from specific graphs, either seeds or generated examples.

Subset/superset conjectures are based upon both seeds and definitions. For any pair of properties p and q, GT seeks:

- similarity in the seed sets for p and q (in decreasing order of significance: equal sets, one a subset of the other, a non-null intersection)
- seeds of property p which are known to have the property q
- similarity between the operators for p and q (i.e., which primitives are employed and in what groupings)

The strong focus on seeds is justified both by their role as prototypes and by efficiency; seeds tend to be small and few in number.

Before GT's heuristics explore the third mathematical question, the equivalence of p and q, they require that the two associated subsumptions have been either proved or conjectured. Alternative definitions (*characterizations*) of classes are common in mathematics because they support conjecture and, therefore, research. GT demonstrates such use of alternative definitions. Consider, for example, the class of graphs known as chains. (A *chain* is a connected graph in which two nodes have degree one and all others have degree two.) GT has two different definitions of chain. Based on the operators, one suggests that a chain may be a cycle, and the other suggests that a chain may be a tree. GT formulates and investigates both conjectures, and discovers that the first is incorrect and the second correct.

Conjectures about disjointness are really conjectures that a merger will fail. Thus a conjecture in GT about the disjointness of p and q is expressed as a plan to merge p and q. If the seed sets for p and q are disjoint, the possibility of the disjointness of P and Q will be conjectured in the form of a plan to attempt the merger of p and q.

5. Summary of Results and Future Work

According to Michalski's characterization of learning systems[Mic86], GT learns both by observation (of its input examples and definitions) and by discovery (upon construction of new examples and properties). GT inductively infers conjectures from examples and definitions, and also proves deductively from the same definitions. Figure 1 displays the ACYCLIC frame both before and after one of GT's runs. No specific tasks were input, only the general directive to explore the knowledge base. GT formulated its own conjectures and then attempted to construct proofs for them based on the structure of the definitions. The modifications to the representation for ACYCLIC constitute learning as defined in [Mic86]. Clearly GT learns how ACYCLIC relates to other concepts and constructs and stores additional examples of acyclic graphs. GT learns about graph theory by conjecturing and exploring simple relations among graph properties.

GT is able to conjecture theorems in graph theory. Conjecture is driven by extremal examples and definitions. Example-driven discovery is based upon prototypical graphs (seeds) which are extremal cases of individual properties and therefore likely to be rich in associations. Definition-driven discovery focuses upon the transformations which change one graph with a property into another graph with the same property. The requirement that a definition be complete effectively limits such transformations to minimal changes. (For example, a connected graph may be transformed by adding a new vertex with one edge to an old vertex. Requiring that the new vertex be connected to more than one old vertex would create a different, more restricted, set of graphs.) The minimality of these changes and the limited vocabulary of operator primitives makes relations between the transformations in the definitions more readily apparent.

GT is able to prove theorems in graph theory which it has conjectured. Proofs rely heavily on a procedure to test for subsumption and a procedure for merger to represent graphs with more than one property. Running on a Symbolics 3675 in Symbolics Common Lisp, GT successfully conjectures and proves, among other theorems, the following:

- Every tree is acyclic.
- Every tree is connected.
- The set of acyclic, connected graphs is precisely the set of trees.
- There are no odd-regular graphs on an odd number of vertices.

Although GT is described as domain-specific, it offers domain-independent lessons as well. The richness of the semantic network GT constructs is due to extensive exploration. Thus, rather than a burden, exhaustive search is one of GT's strengths. In the META-DENDRAL tradition, GT can afford exhaustive search because its representation is highly-controlled. The design of the language for graph property definitions engineers GT for success, because it capitalizes on the inherent similarities within object classes and captures the commonalities underlying class definitions.

Plans for GT's future development are based upon the power and flexibility of the p-generator representation. Within the discovery framework described here, plans exist to extend the p-generator language for the representation of directed graphs and, eventually for labelled graphs. These extensions will also provide a testbed for the study of performance under representational shifts. Work is under way to use additional example-based reasoning, particularly counterexamples, to evaluate the agenda and guide search. Thus discovery will derive additional data-driven support, while maintaining its theory-driven component. GT's knowledge base will be expanded with more concepts gleaned from the benchmark texts. Mathematicians studying interesting sets of graph properties are invited to submit them to GT. The shell of GT is a domain-independent research tool for recursive property description. Applications of this shell to mathematical domains other than graph theory are currently under study.

Acknowledgements: The author thanks N.S. Sridharan and Virginia Teller for their assistance and support in this research and its presentation.

REFERENCES

[Bon]Bondy,J. and Murty,U., *Graph Theory with Applications*, New York, North-Holland, 1976.

[Eps83]Epstein,S.L.,"Knowledge Representation in Mathematics: A Case Study in Graph Theory," Ph.D. dissertation, Rutgers University, 1983.

[Eps87]Epstein,S.L., "Languages for Problem Solving in Graph Theory," in *The Role of Language in Problem Solving 2*", J.C. Boudreaux, B.W. Hamill and R.N. Jernigan, eds., North-Holland, 1987, pp.261-300.

[Eps]Epstein,S.L., "Learning and Discovery - One System's Search for Mathematical Knowledge," in preparation.

[Har]Harary,F., *Graph Theory*, Reading, MA, Addison-Wesley, 1972.

[Len76]Lenat,D.B., "AM: An Artificial Intelligence Approach to Discovery in Mathematics," Ph.D. dissertation, Stanford University, 1976.

[Len84]Lenat,D.B., "Why AM and EURISKO Appear to Work," *Artificial Intelligence*, Volume 23, Number 3, August 1984, pp.269-294.

[Mic86]Michalski,R.S., "Understanding the Nature of Learning: Issues and Research Directions," in *Machine Learning: An Artificial Intelligence Approach*, Volume II, R.S. Michalski, J.G. Carbonell and T.M. Mitchell, eds., Palo Alto, Tioga, 1986, pp.3-25.

[Mic83]Michalski,R.S. and R.E. Stepp, "Learning from Observation: Conceptual Clustering," in *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell and T.M. Mitchell, eds., Palo Alto, Tioga 1983, pp.331-363.

[Michen]Michener,E.R., "Understanding Understanding Mathematics," Technical Report AI MEMO-488, MIT, August, 1978.

[Ore]Ore,O., American Mathematical Society Colloquium Publications, Volume 38: *Theory of Graphs*, Providence, RI, American Mathematical Society, 1962.

[Rit]Ritchie,G.D. and F.K. Hanna, "AM: A Case Study in AI Methodology," *Artificial Intelligence*, Volume 23, Number 3, August 1984, pp.269-294.