Noname manuscript No.
(will be inserted by the editor)

Automated Conjecturing III

Property-relations Conjectures

C. E. Larson · N. Van Cleemput

Received: date / Accepted: date

Abstract Discovery in mathematics is a prototypical intelligent behavior, and an early and continuing goal of artificial intelligence research. We present a heuristic for producing mathematical conjectures of a certain typical form and demonstrate its utility. Our program conjectures relations that hold between properties of objects (property-relation conjectures). These *objects* can be of a wide variety of types. The statements are true for all objects known to the program, and are the simplest statements which are true of all these objects.

The examples here include new conjectures for the hamiltonicity of a graph, a well-studied property of graphs. While our motivation and experiments have been to produce mathematical conjectures—and to contribute to mathematical research—other kinds of interesting property-relation conjectures can be imagined, and this research may be more generally applicable to the development of intelligent machinery.

Keywords Automated conjecturing · Automated mathematical discovery · Property-relations conjectures

1 Introduction

Discovery in mathematics is a prototypical intelligent behavior, and an early goal of artificial intelligence research. While substantial effort has gone into research on

C. E. Larson
Department of Mathematics and Applied Mathematics
Virginia Commonwealth University
Richmond, VA 23284 USA
Tel.: +1-804-828-5576
Fax: +1-804-828-8785
E-mail: clarson@vcu.edu

N. Van Cleemput
Department of Applied Mathematics, Computer Science and Statistics
Ghent University
Krijgslaan 281 - S9 - WE02
9000 Ghent, Belgium

automating mathematical discovery, this research has lagged the successes of other areas of artificial intelligence research; and, furthermore, the large and growing body of research on machine learning has contributed almost nothing to the limited successes in the development of programs that automate mathematical discovery.

One, of several, activities that are necessary for the discovery of new mathematical facts is the formation of a *conjecture*. The problem addressed here is how to produce mathematical conjectures which take the form of a relation between properties of a mathematical object (property-relation conjectures). We present a heuristic for producing property-relation conjectures and examples of the success of this heuristic. While our heuristic idea is not new—it was outlined in the first paper of this series—we have now implemented it and demonstrated that it works.

The Chvátal-Erdös Theorem [9] is a prototypical example of a property-relation statement: if the independence number of a graph is no more than its vertex-connectivity then the graph has a hamiltonian cycle. The independence number and connectivity numbers are graph *invariants*—numbers associated with graphs (the *independence number* is the size of a largest set of vertices which do not have an edge between any pair, and the *connectivity number* is the fewest vertices which must be removed in order to disconnect the graph). Here, "independence number is no more than its connectivity number" defines a graph property: for any particular graph, its independence number is either less than or equal to its connectivity number, or it is not and its independence number is greater than its connectivity number. A hamiltonian cycle in a graph is a cycle which visits each vertex exactly once. So "has a hamiltonian cycle" is a graph property: either a graph has a hamiltonian cycle or it does not. The Chvátal-Erdös Theorem states a *relation* between graph properties: that a graph has one graph property implies that it has another.

Another example of a property-relation statement is the following open conjecture of our program, discussed below: if the radius of a graph equals its diameter and the graph is eulerian then the graph is hamiltonian. The *radius* of a graph is the minimum distance of any of its vertices to the most distant vertex from it), and the *diameter* is the maximum distance between any pair of vertices. The radius and the diameter are both graph invariants, while "having equal radius and diameter" is a graph property. The form of this conjecture is also a *conditional* (or if-then) statement, asserting a relationship between graph properties.

There are two issues in the production of mathematical conjectures. The first issue is to produce syntactically correct mathematical statements: this is relatively easy and can be done recursively in terms of atomic propositions and any collection of propositional operators. The second issue—and real difficulty—is to produce statements which are of mathematical interest. While the "interest" of a statement might be taken in a psychological sense, for the purposes of contributing to scientific research, we take it to mean just that: the statement, if true, should advance mathematical research. In fact, this can be formulated *objectively*: the statement, if true, should say something about a problem that is already being researched and which is not implied by the existing published corpus of theorems.

These mathematical *objects* in the conjectures produced by our program can be of a wide variety of types. The produced statements are true for all objects known to the program, and are the simplest statements which are true of all these objects. The program can also be provided with any existing theoretical knowledge and, in this case, the program is guaranteed to produced statements which are not implied

by this theory. The examples here include new conjectures for the hamiltonicity of a graph, a well-studied property of graphs.

While our motivation and experiments have been to produce mathematical conjectures—and to contribute to mathematical research—other kinds of interesting property-relation conjectures can be imagined, and this research may be more generally applicable to the development of intelligent machinery. A referee suggests we add a concrete example of what this claim might mean.

Chomp is a two-player perfect information impartial game, invented by David Gale in the 1970s [23]. Every Chomp position either has a winning strategy for the *n*ext player to play (an N-position) or for the player that did *p*reviously play (a P-position). We are primarily interested in developing the theory and identification of P-positions. The reason is that, when a player makes a move, her move results in a board position for her opponent. If she has a winning strategy and plays perfectly she will give her opponent a P-position. So her perfect play boils down to identifying the P-positions that are reachable from her current position.

Together with students in a summer research project we have used the Conjecturing program to develop a theory of necessary and sufficient conditions for Chomp P-positions. This "theory" (consisting mostly of unproved conjectured statements) can be used for deciding on a move: one heuristic is to choose the first considered move that is consistent with the theory (and otherwise choose any move which is maximally consistent). Successful play would constitute intelligent behavior. While this is work-in-progress it suggests one way ideas used here more generating mathematical conjectures may be more generally applied to making intelligent decisions between possible actions.

2 Background & Previous Work

This research extends the authors' program Conjecturing, which produces invariant-relation conjectures, and is based on a heuristic of Fajtlowicz [21]. The invariant-relations program and related experiments are described in [28]. The user of this program may input example objects of any type, choose invariants (numbers that can be computed from the objects, specified as functions) that may appear in the conjecture statements, choose a specific invariant that will appear on the left-hand side of the conjecture, and choose the form of the inequality: either upper bounds or lower bounds for the chosen invariant. (We use "type" here in a way consistent with computer science sense: practically speaking, the "objects" belong to Python classes—any non-trivial class will have *methods* with numeric or boolean outputs.)

The reported conjectures came from the domains of graph theory, matrix theory, number theory, and combinatorial game theory. Our program is open-source, and operates in Sage (a free and growing mathematical computing environment, similar to Maple, Matlab and Mathematica). The program, examples, and set-up instructions are available at: `http://nvcleemp.github.io/conjecturing/`

In 1948 Turing suggested designing machines to do mathematical research as a starting point towards the design of generally intelligent machines [35]. In the 1950s Newell and Simon developed the Logic Theorist program, the first mathematical theorem-proving program of any kind [34]. The automation of theorem proving is the largest and best-developed area of automated mathematical discovery research. The 1996 computer proof of the Robbins Conjecture [32] was a milestone in this

area. Nevertheless, subsequent success has been limited: no other automated proofs of conjectures of this stature have followed.

In the late-1950s Wang initiated research on automated mathematical conjecturing [37]. A variety of programs have since been developed that either attempt to simulate how research mathematicians make conjectures, or that try to produce conjectures of interest to mathematicians, or both. These include the programs of Fajtlowicz and DeLaVina, as well as Lenat's AM [29–31,16] Epstein's GT [19, 20] Colton's HR [11–13,10], Hansen and Caporossi's AGX [6,7,1], and Mélot's GRAPHEDRON [33,8]. GRAFFITI, GRAFFITI.PC and AGX have led to an especially large number of publications by mathematical researchers. There is also related and interesting work on the automation of mathematical discovery by many others including the GRAPH program of Cvetkovíc and a large group of University of Belgrade collaborators [15], Brigham and Dutton's INGRID program [5,18], the geometry programs of Bagai and collaborators [2,3], the hypergeometric series work of Wilf and Zeilberger [38], and applications of automatic recognition of integer-relations of Borwein, Bailey and their collaborators [4].

Fajtlowicz's GRAFFITI program produced a number of invariant-relations conjectures in graph theory, and was the first program to make research conjectures in mathematics. Our general-use CONJECTURING program was based on his Dalmatian heuristic. Some of GRAFFITI's best known conjectures are bounds for the independence number of a graph. The *residue* of a graph (the number of zeros remaining after repeated application of the Havel-Hakimi procedure to the degree sequence of the graph) is a graph invariant. GRAFFITI conjectured that the residue of a graph is no more than the independence number of the graph. This statement is an inequality where both sides of the inequality are either basic graph invariants or functions of these, a prototypical invariant-relation conjecture. This conjecture was originally proved by Favaron, Mahéo and Saclé [22], and has since been reproved in the literature more than once (see also [26]).

Fajtlowicz's Dalmatian heuristic comprises a truth-test and a significance test. Both GRAFFITI and CONJECTURING produce inequalities between algebraic relations of the input invariants. These are then checked to be true for all examples that are provided to the program. This is the *truth test*. If a produced statement is false for an input object, the statement is rejected as a potential conjecture. Each statement is then tested for *significance* with respect to the input objects and the database of previously produced conjectures. A statement is "significant" if it is not implied by the totality of previously made conjectures: more concretely, a statement is significant if there is at least one input object such that the statement gives a better bound for the user-input invariant than any previously produced conjecture. While *significance* is precisely defined in the next section, and a careful example is presented in [28] for invariant-relations statements, it is worth adding something to this informal description: if sufficient condition conjectures are generated for a property Q, a proposed sufficient condition ("lower bound") P is significant with respect to existing conjectured lower bounds $C_1, \ldots, C_k$, if there is an object which does not have any of the properties $C_1, \ldots, C_k$, but which does have property P.

By the design of the program, each produced conjecture is then "significant" with respect to the previously produced conjectures. Furthermore, if no-longer-significant conjectures are removed whenever significant conjectures are added to

the conjectures store, the number of conjectures (of any particular form) cannot exceed the number of example objects.

These same ideas are generalized and reproduced in our properties-relations program. In this case, instead of inequalities (a type of invariant-relation) the program produces conditional statments (implications, if-then statements). If the user-specified property follows the *if* (is the antecedent of the conditional), the produced statements will give necessary conditions for the given property—these are analogous to the "upper bound" conjectures of our invariant-relations program. If the user-specified property follows the *then* (is the consequent of the conditional), the produced statements will give sufficient conditions for the given property—these are analogous to the "lower bound" conjectures of our invariant-relations program.

The first author described a Dalmatian-style necessary condition heuristic in [27]. DeLaVina and Waller described and implemented a Dalmatian-style sufficient condition heuristic for graph theory conjectures in [17] that they call Sophie. The Sophie version of GRAFFITI.PC has produced some useful conjectures including the following: if the independence number of a graph equals its radius then the graph has a hamiltonian path; this was proved in [17]. A *hamiltonian path* in a graph is a path which visits each vertex exactly once, but is not necessarily a cycle. Here the objects are graphs, and the properties are "has equal independence number and radius" and "has a hamiltonian path". Sufficient conditions for a graph having a hamiltonian path have been of continuing interest [24, 25].

2.1 Dalmatian Heuristic for Properties

We have successfully implemented the heuristic previously described as "future work" in [28]. The main purpose of this paper is to provide examples that demonstrate the success of this idea. That description of our heuristic is reproduced here in order to make this paper self-contained.

The analogues of upper or lower bounds for an invariant of interest are necessary or sufficient conditions for a property of interest. Let P be the property that an integer is perfect. If sufficient conditions for an integer to have this property are desired, a conjecture-making program would need to produce property-expressions $Q_1, Q_2, \ldots$, and statements of the form, "If an integer has property Q_i then it has property P" (or, more simply, "If Q_i then P"). If necessary conditions are desired then the program would need to produce statements of the form, "If an integer has property P then it has property Q_i."

Let $\mathcal{O}_1, \ldots, \mathcal{O}_n$ be examples of objects of a given type. Let $Q_1, \ldots, Q_k$ be properties. And let P be a property for which conjectured necessary or sufficient conditions are of interest. If the objects are the integers $G_1, \ldots, G_n$, and P is the property "is perfect" then $P(G_i)$ would be True if G_i is perfect and False if G_i is not perfect.

An unlimited stream of boolean functions of the properties can then be formed: $Q_1 \wedge Q_2$, $\neg Q_1$, $Q_1 \vee Q_3$, $(Q_2 \wedge Q_4) \vee Q_3$, etc. This stream can be produced in any way at all. In fact they are produced systematically, naively, and completely. The user-given properties are bounds of complexity 1. Unary-function of these are bounds of complexity 2, while binary functions of these are bounds of complexity 3. In general, a unary function of a previously generated bound of complexity k

has complexity $k+1$, while a binary function of previously generated bounds of complexity k and l has complexity $k+l+1$. We later make claims about the "simplicity" of a conjectured bound for P: we say that one conjectured bound is *simpler* than another if it is is less complex (if it has smaller complexity in this recursive definition). In fact our program recursively generates all possible bounds of increasingly higher complexity. No heuristics are used to attempt to intelligently prune this list of expressions prior to their use by the truth and significance tests.

These expressions can then be used to form conjectured necessary or sufficient conditions for P. If we are interested in necessary conditions for P, say, we can form the statements $P \Rightarrow Q_1 \wedge Q_2$, $P \Rightarrow \neg Q_1$, $P \Rightarrow Q_1 \vee Q_3$, $P \Rightarrow (Q_2 \wedge Q_4) \vee Q_3$, etc. These statements can be interpreted as being true for all the objects of the given type. That is, the statement $P \Rightarrow Q_1 \wedge Q_2$ can be interpreted as, "For every object $\mathcal{O}$, $P(\mathcal{O}) \Rightarrow Q_1(\mathcal{O}) \wedge Q_2(\mathcal{O})$." A conjectured necessary condition Q is only added to the database of conjectures if the property passes the following two tests.

1. (*Truth test*). The candidate conjecture $P \Rightarrow Q$ is true for all of the stored objects $\mathcal{O}_1, \ldots, \mathcal{O}_n$, and
2. (*Significance test.*) There is an object $\mathcal{O} \in \{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$ such that $\neg Q(\mathcal{O}) \wedge (C_1(\mathcal{O}) \wedge \ldots \wedge C_r(\mathcal{O}))$, where $C_1, \ldots, C_r$ are the currently stored conjectures. That is, the candidate conjecture would give a better necessary condition for P than any previously conjectured necessary condition.

If we are interested in sufficient conditions for P we can form the statements $Q_1 \wedge Q_2 \Rightarrow P$, $\neg Q_1 \Rightarrow P$, $Q_1 \vee Q_3 \Rightarrow P$, $(Q_2 \wedge Q_4) \vee Q_3 \Rightarrow P$, etc. These statements can be interpreted as being true for all the objects of the given type. That is, the statement $Q_1 \wedge Q_2 \Rightarrow P$ can be interpreted as, "For every object $\mathcal{O}$, $Q_1(\mathcal{O}) \wedge Q_2(\mathcal{O}) \Rightarrow P(\mathcal{O})$." A conjectured sufficient condition Q is only added to the database of conjectures if the property passes the *Truth* and *Significance* tests. In this case the significance test would be as follows: Check that there is an object $\mathcal{O} \in \{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$ such that $Q(\mathcal{O}) \wedge \neg(C_1(\mathcal{O}) \vee \ldots \vee C_r(\mathcal{O}))$, where $C_1, \ldots, C_r$ are the currently stored conjectures. That is, the candidate conjecture would give a better sufficient condition for P than any previously conjectured sufficient condition.

Another way to think about property-relation conjectures is in terms of the sets of objects that have some property. Let $\mathcal{P}$ be the set of objects that have property P. Necessary conditions for membership in $\mathcal{P}$ define a super-class $\mathcal{N}$ of $\mathcal{P}$. What is wanted are conjectures that make this super-class smaller and smaller. So a conjectured necessary condition Q is informative if, together with the previous conjectures, it defines a smaller super-class $\mathcal{N}'$ with $\mathcal{P} \subseteq N' \subset \mathcal{N}$.

Similarly, sufficient conditions for membership in $\mathcal{P}$ define a sub-class $\mathcal{S}$ of $\mathcal{P}$. What is wanted here are conjectures that make this sub-class larger and larger. So a conjectured sufficient condition Q is informative if, together with the previous conjectures, it defines a larger sub-class $\mathcal{S}'$ with $\mathcal{S} \subset \mathcal{S}' \subseteq \mathcal{P}$.

3 Examples

We now give examples of conjectures produced by our program; they address two existing research areas in graph theory. There is nothing particular about graphs for our program. It does happen to be the case that the authors are graph

theorists and have spent substantially more time coding graph theoretic invariants and properties than invariants and properties for any other domain. Also, for this reason, the authors are more familiar with open questions in graph theory than in other mathematical domains.

The form of the conjectures are unquantified conditional statements; these must then be interpreted as statements quantified over some domain. The obvious domain is the *type* of the objects. Nevertheless other more restrictive domains are also possible—and the user may have had a specific domain in mind. Thus the interpretation of the produced conjecture statements may vary. While the input objects may all be *graphs* for instance, if these graphs all happen to be *connected*, then the conjecture statement might variously be interpreted as being about either graphs, connected graphs, or some other class of graphs.

A user of our program, must supply three kinds of *inputs* to the program:

1. A list of objects. The type is arbitrary but to get meaningful results they will all represent the same mathematical type of object. For instance, if you want to generate conjectures about graphs, and `c5`, `k5` and `petersen` are pre-defined graph objects, you would define `objects` = [`c5`, `k5`, `petersen`], and give `objects` as a parameter to the program.
2. A list of properties. These must be functions that are defined for the type of objects in the `objects` list. For instance, if `is_hamiltonian` and `is_even_hole_free` are pre-defined boolean-valued graph functions, you would define `properties` = [`is_hamiltonian`, `is_even_hole_free`] and give `properties` as a parameter to the program.
3. A positive integer listing the position of the invariant in the list of `properties` that you would like to conjecture bounds for from the list of `properties`. For instance if conjectures for the hamiltonicity of a graph are desired, the user would enter 0 in the list of parameters to the program, or set the input parameter `property` (below) to 0.
4. A fourth kind of input is optional: a list of known necessary or sufficient conditions (properties, dependent on conjecture type) which the produced conjectures must improve on, at least for a single object; that is, any produced conjecture must neither be implied by the totality of the previous conjectures together with these additionally listed properties. For instance, in order for a graph to be hamiltonian, the graph must necessarily be connected (that is, there must be a path between any pair of vertices). If necessary condition conjectures for hamiltonicity are desired, the user would define `theory` = [`is_connected`] as a parameter to the program.

Here is the simplest example of the commands needed in order to generate conjectures. This program defines three variables and inputs them into the properties-relation conjecturing function `propertyBasedConjecture`. Here we have only two properties, and only one example object; these are defined by built-in Sage functions. The default is to generate sufficient conditions for the specified property. To generate necessary conditions, the user would add the parameter `sufficient = False` to this function call. And to add known theory stored as the list named `theory`, the user would add the parameter `theory = theory`.

```
properties = [Graph.is_hamiltonian,Graph.is_connected]
property =  properties.index(Graph.is_hamiltonian)
```

```
objects = [graphs.CompleteGraph(3)]
propertyBasedConjecture(objects, properties, property)
```

The *sentential connectives* or *propositional operators* that are built-in to the program and which may appear in the produced conjectures are: "`->`", "`|`" , "`^`", and "&" are binary operators which represent implication, disjunction (inclusive-or), xor (exclusive-or), and conjunction (and), respectively; "`~`" is a unary operator which represents negation.

3.1 Graph Hamiltonicity

Determining whether a graph is hamiltonian is NP-hard; this is one reason research on necessary or sufficient conditions for hamiltonicity has been of continuing interest: Gould's survey papers list a few hundred references [24,25].

We first experimented with generating sufficient conditions for graph hamiltonicity. The example objects given to the program were the complete graphs on three and four vertices (`k3` and `k4`), the path on four vertices `p4`, and the Petersen graph; the two complete graphs are hamiltonian, while the other two graphs are not. The properties given to the program included all built-in Sage graph properties, together with a few properties we coded. The first run produced a single conjecture:

```
(is_eulerian) -> (is_hamiltonian)
```

We interpret this conjecture as: for every graph, if the graph is *eulerian* then the graph is hamiltonian. A graph is eulerian is there is a cycle which contains all of the edges of the graph; of the four input graphs, only `k3` is eulerian: `k3` is the only graph which satisfies the sufficient condition. In order then for this conjecture to be true of all input objects, it must only be true for `k3` and `k3` is indeed hamiltonian. The program could potentially have made one more conjecture in order to account for the hamiltonicity of `k4`, but it did not find one before it timed out. The conjecture is false; it is easy to find counterexamples: for example, the graph formed by identifying a single vertex from each of two 4-cycles is eulerian but not hamiltonian.

Now it is possible to take all coded graphs as the initial input of objects to the conjecturing program. There will be a single run—with potentially as many conjectures as input objects—all of which are true for these objects. Another way to use the program—and the way that we have done in our experiments—is to begin with only a few (four) input objects and add an object to the input list *only if* it is a counterexample to some previously generated conjecture. These counterexamples may be constructed by the user or may be found by systematic search through all small graphs. The advantage to this iterative use of the program is that, since only significant examples are added to the input list, and the number of generated conjectures cannot be more than the number of input objects, the number of conjectures is limited—researchers generally prefer fewer rather than more conjectures—and also they tend to be less complex—there are fewer objects that each conjecture must satisfy.

There is also something distinctly human in this approach: humans focus on a limited number of examples which have previously made some impression—rather than on comprehensive catalogs of examples.

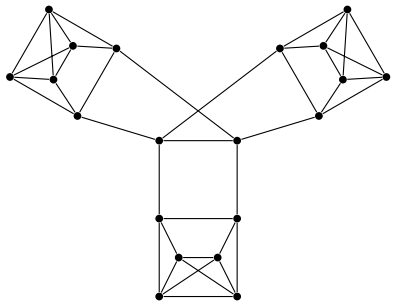

Fig. 1 Gould's counterexample: an eulerian, regular, two-connected graph which is not hamiltonian

We report now on a few more conjectures from the subsequent runs in this experiment. The third run produced just the following two conjectures:

```
(is_clique) -> (is_hamiltonian)
(is_connected_dirac) -> (is_hamiltonian)
```

A graph is a *clique* if every pair of vertices are adjacent; these graphs are hamiltonian. A connected graph is *dirac* if the graph has n vertices and each vertex is adjacent to at least $\frac{n}{2}$ others. Dirac's Theorem says that these graphs are hamiltonian. So these two conjectures are both true.

After a few rounds the program made the following conjecture.

```
(((is_eulerian) & (is_regular)) & (is_two_connected)) -> (is_hamiltonian)
```

A graph is *regular* if each vertex is adjacent to the same number of other vertices. It is *two-connected* if at least two vertices must be removed in order to disconnect the graph. Gould found a 20 vertex counterexample (Fig. 1) to this conjecture: it is eulerian, regular, two-connected but not hamiltonian. The last conjecture of this run of sufficient condition conjectures for graph hamiltonicity is:

```
((has_radius_equal_diameter) & (is_eulerian)) -> (is_hamiltonian)
```

The authors do not know a counterexample.

Finally we report generated necessary condition conjectures for hamiltonicity. Interestingly, only a handful of necessary conditions appear in the literature; these are far exceeded by the number of published sufficient conditions. One obvious necessary condition is that the graph be two-connected. Another, due to Van Den Heuvel, is defined in terms of the graph's eigenvalues [36]. Both of these appeared as conjectures of our program. But, since we were interested in conjectures that potentially advanced beyond what researchers already know, we added these to the `theory` parameter, thereby requiring the produced conjectures to not be consequences of these known bounds: so every conjectured necessary condition had to be false for at least one input graph which was two-connected and for at least one input graph which had the Van Den Heuvel property. Here is the complete list of conjectures from the last round:

```
(is_hamiltonian) -> (((is_planar) & (is_regular)) -> (is_anti_tutte2))
(is_hamiltonian) -> ((is_class1) | (has_radius_equal_diameter))
(is_hamiltonian) -> (((is_van_den_heuvel) -> (is_overfull))^(is_class1))
(is_hamiltonian) -> (((is_cubic) -> (is_planar)) | (is_bipartite))
```

The *degree* of a vertex is the number of edges adjacent to that vertex. The *chromatic number* is the fewest number of colors required to assign different colors to adjacent vertices. A graph is *planar* if it can be drawn on the plane without crossed edges; *class1* if the chromatic number is no more than the maximum degree of any vertex; *overfull* if the number n of vertices is odd and twice the number of edges is more than n-1 times the maximum degree; *cubic* if every vertex is adjacent to exactly three others; and *bipartite* if the vertices can be partitioned into two independent sets. The *domination number* of a graph is the fewest number of vertices so that each vertex is either in this collection or adjacent to one of these; then a graph is `anti_tutte2` if it is connected and its independence number is no more than one less than the sum of its domination number and radius. These conjectures are open: we do not know counterexamples for any of these.

3.2 Even Hole-Free Graphs

A *hole* in a graph is a cycle with more than three vertices; and an *anti-hole* is the complement of a hole. A graph is *perfect* if the chromatic number of every subgraph equals the size of a largest clique. The very well-known Strong Perfect Graph Theorem is stated in terms of non-existence of subgraphs which are odd holes ("odd hole-free") or odd anti-holes ("odd anti-hole-free"). This research in turn inspired research into characterizing graphs which are "even hole-free" [14]. Both necessary and sufficient conditions for graphs to have this property are of interest to researchers. The following conjecture was from the initial run of sufficient condition conjectures.

```
(is_chordal) -> (is_even_hole_free)
```

A graph is *chordal* if no induced cycle has more than three vertices. This conjecture is true. `is_chordal` was then added to the `theory` variable. Here are open sufficient condition conjectures.

```
(~(is_two_connected)) & (has_residue_equals_alpha)) -> (is_even_hole_free)
((~(is_eulerian)) & (diameter_equals_twice_radius)) -> (is_even_hole_free)
((~(is_claw_free)) & (has_residue_equals_alpha)) -> (is_even_hole_free)
(((is_circular_planar) -> (is_perfect)) -> (is_clique)) -> (is_even_hole_free)
```

A *claw* in a graph is an induced subgraph consisting of a vertex adjacent to three independent vertices; a graph is *claw-free* if it does not have a claw.

4 Discussion

The main purpose of this paper was to demonstrate the effectiveness of Fajtlowicz's Dalmatian heuristic for the generation of property-relation conjectures in mathematics. We have produced a working (and open source) program and demonstrated its utility for producing conjectures that advance research on two different graph theory questions. Our program extends what can be done with automated mathematical discovery programs.

Several points are worth mentioning. As with our invariant-relations conjecturing program, this program is domain independent: if the program is given matrix

examples and matrix properties, for instance, it would produce necessary or sufficient conditions for a specified matrix property.

Also no "expert knowledge" is required to prepare the program for use in a domain. That is, the "knowledge" required for the program to run is some object examples and some suitable properties. These could be found in textbooks for a specific domain. We expect that a good undergraduate mathematics major could code in all the properties, objects, and theoretical bounds found in an undergraduate text. The undergraduate of course must have some "mathematical maturity" and would be more expert than say an English major, but wouldn't have necessarily published any mathematical papers or have the comprehensive expertise of her teachers.

Relatedly, all published properties, objects, and theorems in a given mathematical domain could be coded in. This would require substantial and collaborative effort—but may have significant payoff. We have begun doing this ourselves in the case of graph theory. In this case, it would not be possible for any expert to perform "better" than our program. That is, the expert may aim to produce conjectured statements using any known (published) properties that are true for all known examples and are not implied by the known theory. The program by design produces the least "complex" statement (by any natural measure—for instance the number of atomic properties that appear in the statement) that is true for all known examples and is not implied by the input theory. At least the expert could not produce a simpler conjecture.

It should also be emphasized that, as with all successful automated mathematical discovery programs, *success is by design.* This program together with every other program that contributes to mathematical research was *designed* to contribute to mathematical research. In this case, it was because mathematicians are looking, for instance, for new necessary and sufficient conditions for the hamiltonicity of a graph and our program was designed to produce this kind of conjecture. This is in contrast with programs like Lenat's AM. Lenat's program reproduced several well-known mathematical conjectures, for instance Goldbach's conjecture that every even integer is the sum of two primes—but it did not make new conjectures, nothing that would advance mathematics. Goldbach's conjecture might be described as a conjecture about representations of integers by sums. Lenat's program was not designed to make conjectures about representations of integers by sums—and, thus, there was no reason to expect it to make a contribution to this research.

There are several things that can be done to improve the success of our program's utility to domain scientists. We mentioned one: to code all known properties, key examples and theorems in the given domain. Here are three more ideas that can improve the utility of our program:

1. The conjectures our program produces are the simplest statements using the given ingredients that are true for all provided objects. Nevertheless these statements may be quite complicated. A mathematician, like any other human, can have a difficult time comprehending complex statements. In that case, they are in danger of losing their utility. It is often useful to find restricted domains where simpler statements are true. It is not clear how to automate finding a suitable domain that allows for simple form (or scannable, or comprehendible) conjectures.

2. The program requires example objects as input. The produced conjectures are true of all of these objects. If all published examples were included as inputs, the program would produce conjectures that are true for all of these. Nevertheless, the conjectures may be false. To find counterexamples to false conjectures then would necessarily expand knowledge in the investigated domain. One approach, which we have experimented with, is to systematically produce objects from the simplest to the more complex. For instance, for graphs one could first generate all two vertex graphs, then all three vertex graphs, etc. This can be done with more or less sophistication. Nevertheless, only graphs of very small size can be produced: the numbers of possible graphs grows exponentially. The case of most other interesting mathematical objects is likely similar. This means that only small sized counterexamples could be found by systematic generation. Of course counterexamples may be of larger sizes. A better approach would be intelligent generation of potential counterexamples to conjectures. We do not know how to do this—but it would be a substantial contribution to research on automated mathematical discovery.
3. It should be possible and useful for invariant-relation and property-relation programs to interact. We know how to use invariants to define properties, and how to use properties to define invariants; but we do not know how to make these two programs interact systematically in a way that helps the user advance her mathematical goals.

Finally, we see conjecturing—and conjecture-revision in the face of contradictory data (counterexamples)—as a central feature of intelligence. There is a growing body of evidence, especially from visual perception, that we are constantly making conjectures about our world based on incomplete perceptual data. Intriguing research on split-brain patients demonstrates that the linguistic brain will generate (conjecture) stories about the limited data available to it. One side of the brain for instance may only know details, not the "whole story", but, as the halves cannot communicate, the other half can only guess at the missing details—and does so.

It is worth emphasizing then, that our program is generalizable to non-mathematical property-relation statements. The ingredients are the same: example objects and properties of those objects. The only requirement is that the object-types have computable properties. We believe and hope that quick on-the-fly conjecture-based heuristics might be used in the design of machines that perform a variety of intelligent behaviors.

Acknowledgements

The authors are grateful to the referees, whose comments helped us clarify and improve this paper.

References

1. M. Aouchiche, G. Caporossi, P. Hansen, and M. Laffay. Autographix: a survey. *Electronic Notes in Discrete Mathematics*, 22:515–520, 2005.
2. R. Bagai, V. Shanbhogue, J. M. Żytkow, and S.-C. Chou. Automatic theorem generation in plane geometry. In *Methodologies for Intelligent Systems*, pages 415–424. Springer, 1993.

3. R. Bagai, V. Shanbhogue, J. M. Zytkow, and S-C. Chou. Discovery of geometry theorems: avoiding isomorphic situation descriptions. In *Computing and Information, 1993. Proceedings ICCI'93., Fifth International Conference on*, pages 354–358. IEEE, 1993.
4. J. M. Borwein and D. H. Bailey. *Mathematics by experiment: Plausible reasoning in the 21st century*. AK Peters Natick, MA, 2004.
5. R. Brigham and R. Dutton. INGRID: A software tool for extremal graph theory research. *Congr. Numer*, 39:337–352, 1983.
6. G. Caporossi and P. Hansen. Variable neighborhood search for extremal graphs: 1 the autographix system. *Discrete Mathematics*, 212(1):29–44, 2000.
7. G. Caporossi and P. Hansen. Variable neighborhood search for extremal graphs. 5. three ways to automate finding conjectures. *Discrete Mathematics*, 276(1):81–94, 2004.
8. J. Christophe, S. Dewez, J.-P. Doignon, G. Fasbender, P. Grégoire, D. Huygens, M. Labbé, S. Elloumi, H. Mélot, and H. Yaman. Linear inequalities among graph invariants: using GraPHedron to uncover optimal relationships. *Networks*, 52(4):287–298, 2008.
9. V. Chvátal and P. Erdös. A note on hamiltonian circuits. *Discrete Mathematics*, 2(2):111–113, 1972.
10. S. Colton. Refactorable numbers—a machine invention. *Journal of Integer Sequences*, 2(99.1):2, 1999.
11. S. Colton. *Automated theory formation in pure mathematics*. Springer Heidelberg, 2002.
12. S. Colton. Automated conjecture making in number theory using HR, Otter and Maple. *Journal of Symbolic Computation*, 39(5):593–615, 2005.
13. S. Colton, A. Bundy, and T. Walsh. Automatic concept formation in pure mathematics. In *IJCAI'99 Proceedings of the 16th international joint conference on Artificial Intelligence, Volume 2*, pages 786–791. Morgan Kaufmann Publishers, 1999.
14. M. Conforti, G. Cornuéjols, A. Kapoor, and K. Vušković. Even-hole-free graphs part i: Decomposition theorem. *Journal of Graph Theory*, 39(1):6–49, 2002.
15. D. Cvetković and I. Gutman. The computer system GRAPH: A useful tool in chemical graph theory. *Journal of computational chemistry*, 7(5):640–644, 1986.
16. R. Davis and D.B. Lenat. *Knowledge-based systems in Artificial Intelligence*. McGraw-Hill International Book Co., 1982.
17. E. DeLaVina, R. Pepper, and W. Waller. Independence, radius and Hamiltonian paths. *MATCH Commun. Math. Comput. Chem.*, 58(2):481–510, 2007.
18. R. D. Dutton, R. C. Brigham, and F. Gomez. INGRID: A graph invariant manipulator. *Journal of symbolic computation*, 7(2):163–177, 1989.
19. S. L. Epstein. On the discovery of mathematical theorems. In *IJCAI*, pages 194–197, 1987.
20. S. L. Epstein. Learning and discovery: One system's search for mathematical knowledge. *Computational Intelligence*, 4(1):42–53, 1988.
21. S. Fajtlowicz. On conjectures of Graffiti. V. In *Graph Theory, Combinatorics, and Algorithms, Vol. 1, 2 (Kalamazoo, MI, 1992)*, Wiley-Intersci. Publ., pages 367–376. Wiley, New York, 1995.
22. O. Favaron, M. Mahéo, and J.-F. Saclé. On the residue of a graph. *J. Graph Theory*, 15(1):39–64, 1991.
23. D. Gale. A curious Nim-type game. *American Mathematical Monthly*, pages 876–879, 1974.
24. R. J. Gould. Updating the Hamiltonian problem—a survey. *J. Graph Theory*, 15(2):121–157, 1991.
25. R. J. Gould. Advances on the Hamiltonian problem–a survey. *Graphs and Combinatorics*, 19(1):7–52, 2003.
26. Jerrold R. Griggs and Daniel J. Kleitman. Independence and the Havel-Hakimi residue. *Discrete Math.*, 127(1-3):209–212, 1994. Graph theory and applications (Hakone, 1990).
27. C. E. Larson. A survey of research in automated mathematical conjecture-making. In *Graphs and Discovery*, volume 69 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 297–318. Amer. Math. Soc., Providence, RI, 2005.
28. C. E. Larson and N. Van Cleemput. Automated conjecturing I: Fajtlowicz's Dalmatian heuristic revisited. *Artificial Intelligence*, 231:17–38, 2016.
29. D. B. Lenat. The ubiquity of discovery. *Artificial Intelligence*, 9(3):257–285, 1977.
30. D. B. Lenat. On automated scientific theory formation: a case study using the am program. *Machine intelligence*, 9:251–286, 1979.
31. D. B. Lenat. The nature of heuristics. *Artificial Intelligence*, 19(2):189–249, 1982.

32. W. McCune. Solution of the Robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.
33. H. Mélot. Facet defining inequalities among graph invariants: the system GraPHedron. *Discrete Applied Mathematics*, 156(10):1875–1891, 2008.
34. H. A. Simon and A. Newell. Heuristic problem solving: The next advance in operations research. *Operations Research*, 6(1):1–10, 1958.
35. A. Turing. Intelligent machinery. *The Essential Turing*, pages 395–432, 2004.
36. J. Van Den Heuvel. Hamilton cycles and eigenvalues of graphs. *Linear algebra and its applications*, 226:723–730, 1995.
37. H. Wang. Toward mechanical mathematics. *IBM J. Res. Develop.*, 4:2–22, 1960.
38. H. S. Wilf and D. Zeilberger. Towards computerized proofs of identities. *Bulletin of the American Mathematical Society*, 23(1):77–83, 1990.