Automated conjecturing of Frobenius numbers via grammatical evolution

Nikola Adžaga *

Abstract

Conjecturing formulas and other symbolic relations occurs frequently in number theory and combinatorics. If we could automate conjecturing, we could benefit not only from faster conjecturing but also from finding conjectures previously out of our grasp. Grammatical evolution, a genetic programming technique, can be used for automated conjecturing in mathematics. Concretely, this work describes how one can interpret the Frobenius problem as a symbolic regression problem, and then apply grammatical evolution to it. In this manner, a few formulas for Frobenius numbers of specific quadruples were found automatically. The sketch of the proof of one conjectured formula, using lattice point enumeration method, is provided as well.

The same method can easily be used on other problems to speed up and enhance the research process.

1 Introduction

Unlike automation of theorem-proving, automated conjecturing is still a rare topic in mathematics. Probably the most significant example of a program used for automated conjecturing is Fajtlowicz's Graffiti [Fajtlowicz and Waller 1986], which produced hundreds of conjectures in graph theory and inspired dozens of papers (e. g. [Erdős et al. 1995]). Another example is Colton's HR [Colton 2002]. Colton considered mathematics "a new domain for datamining". Using very simple algorithms on existing databases of mathematical knowledge, he discovered, for example, a necessary condition for perfect numbers. Advances in artificial intelligence and the successes achieved by applying artifical intelligence in other sciences suggest that, simply by treating examples and results of math experiments as data for a machine learning problem, we could gain new insights automatically.

*Faculty of Civil Engineering, University of Zagreb, e-mail: nadzaga@grad.hr

One method that could be very useful for mathematics is grammatical evolution (GE). GE is a genetic programming technique (i. e., used for automated programming), and as such, it can easily generate conjectures of deterministic type. On the other hand, GE is a very general method, applicable to any problem which can be described using a context-free grammar. Indeed, it was successfully used in fields as various as computer-aided design ([Fenton et al. 2014]) and computational finance ([O'Neill et al. 2002]).

To author's knowledge, the work described herein is the first instance of using grammatical evolution in mathematics, so this work can be seen as a proof-of-concept demonstration – grammatical evolution can be used for conjecturing in mathematics. We chose to show that for Frobenius problem. It is particularly suitable for this study because it consists of finding formulas, so GE is applied in its most usual form.

Frobenius problem is the following: given relatively prime natural numbers $a_1, \ldots, a_n$, find the *Frobenius number*, the largest natural number that is not representable as a non-negative integer linear combination of $a_1, \ldots, a_n$. This number is denoted as $g(a_1, \ldots, a_n)$.

For $n = 2$, it is long known that $g(a_1, a_2) = a_1a_2 - a_1 - a_2$ (usually attributed to [Sylvester 1882]).

For $n = 3$, the quest for a similar simple formula has been the subject of extensive research. Curtis ([Curtis 1990]) has shown that this quest, in a way, cannot succeed – there is no finite set of polynomials $\{f_1, \ldots, f_k\}$ such that, for each choice of a_1, a_2, a_3, there is some $i \in \{1, \ldots, k\}$ such that $f_i(a_1, a_2, a_3) = g(a_1, a_2, a_3)$. Still, efficient algorithms for calculating Frobenius number have been developed, as well as numerous lower and upper bounds (for an extensive review on these and other results regarding Frobenius number, one can check [Ramírez Alfonsín 2005]).

For a fixed $n \geqslant 4$, there is a polynomial time algorithm, but it is impractical. There is no proven efficient algorithm, but various formulas for some specific cases were found, e. g., for arithmetic and geometric sequences of arbitrary length. For $n = 4$, more specific formulas were found, e. g., in [Dulmage and Mendelsohn 1964] was found that

$$g(a, a+1, a+2, a+4) = (a+1)\left\lfloor \frac{a}{4} \right\rfloor + \left\lfloor \frac{a+1}{4} \right\rfloor + 2\left\lfloor \frac{a+2}{4} \right\rfloor - 1.$$

We show that grammatical evolution can be used for automated conjecturing in mathematics. The rest of this work is divided as follows. The second section is the most important part: it describes how one can interpret the Frobenius problem as a symbolic regression problem, and then apply grammatical evolution to it. In this manner, a few formulas for Frobenius number were conjectured, including the following one:

$$g(3k+1, 3k+4, 6k+3, 6k+9) = (3k+1)\left(k - \left\lfloor \frac{3k+1}{21} \right\rfloor\right) - 1 \quad \forall k \geqslant 5.$$

This section also shows the briefest sketch of the proof of this formula, using lattice point enumeration method. More important is the fact that, simply by changing the

input data, we easily generated conjectures for quadruples of different kind, and even for sextuples. The last section discusses the benefits and limitations of using grammatical evolution as an automated conjecturing method in mathematics.

2 The Frobenius Problem as Symbolic Regression Problem

We interpret the Frobenius problem as a symbolic regression problem, and then approach this problem using grammatical evolution. Grammatical evolution is a method of automatic programming which uses grammars and evolutionary approach. Binary chromosomes are used. *Codons*, groups of 8 bits, represent integer numbers. Codons determine which rule from the grammar will be used.

Following the short background on grammars, we explain how we used grammatical evolution and obtained results.

2.1 Context-free grammars and Backus–Naur form (BNF)

Grammars describing programming languages are defined following the formal languages, which were introduced as a scientific discipline by Noam Chomsky (e. g. [Chomsky 1956]) during his search for simple rules capable of describing natural language. Chomsky concluded that context-free grammars are sufficient for formalizing the grammar of the English language. These grammars form theoretical basis for most programming languages, but the importance of grammars is even greater since they provide the means for formalizing knowledge. If a finite set of simple rules can describe the grammar of English language, then such sets can describe many other phenomena, which in turn opens up a possibility for automated research on these phenomena.

The Backus–Naur form is the most commonly used notation for expressing the grammar of a language. BNF defines syntax rules (as in [Naur 1963]), and it formalizes syntactic expressions. By listing the set of production rules, we completely determine the grammar. In each rule there is exactly one variable on its left hand side, while on its right hand side there are expressions which can replace the given variable.

More formally, context-free grammar can be defined in a following manner: For a given set of symbols A, denote by A^* the set of all sequences of symbols from A (including the empty sequence). Context-free grammar is a quadruple $G = (N, T, P, S)$, where N is the set of nonterminal symbols (which have to be replaced), T is the set of terminal symbols (which cannot be further replaced by other symbols), and P is a finite set of production rules in form $A ::= a$, where $A \in N$ and $a \in (N \cup T)^*$. Finally, $S \in N$ is a start symbol. The example of BNF (the grammar used in this research) is given in the subsection after the next one.

2.2 Interpreting the Problem as Symbolic Regression Problem

We calculated Frobenius numbers for 40 quadruples of the form $(x, x+3, 2x+1, 2x+7)$ with a relatively simple recursive algorithm (using dynamic programming). These quadruples and their Frobenius numbers serve as the input data for grammatical evolution.

The form of these quadruples was chosen as a simple form for which the Frobenius number was previously unknown. However, as we will see later, the same method is easily applied to quadruples of other forms, and even for determining the Frobenius number of more than four integers.

The input data then looks as shown in Table 1.

x	x+3	2x+1	2x+7	Frobenius number (target function)
3	6	7	13	11
4	7	9	15	10
5	8	11	17	14
...	...	...	...	...

Table 1: Input data – Frobenius number for quadruples $(x, x+3, 2x+1, 2x+7)$.

We are trying to find an expression which is equal to $g(x, x+3, 2x+1, 2x+7)$ for all given x. This task can be seen as a symbolic regression problem: its solution is an expression consisting of x and some algebraic operations. Symbolic regression problems can be solved by grammatical evolution.

2.3 Grammatical Evolution

Grammatical evolution is a method of automatic programming that uses grammars and an evolutionary approach. A standard representation of each candidate solution, called an *individual*, is an array of bits (i.e., binary chromosomes are used). Bits are grouped into 8-bit chunks called codons, which represent integer numbers. These codons determine which production rule from BNF will be used. Regardless of their integer value, codons are not skipped, but considered modulo the number of possibilities. For example, given the symbol <op>, say that there are three production rules available to replace it, one to replace it with +, one to replace it with −, and one to replace it with $*$ (multiplication). We denote this as

$$\begin{aligned} \langle \text{op} \rangle ::= \ & + && (0) \\ | \ & - && (1) \\ | \ & * && (2) \end{aligned}$$

If the codon value is greater than 2, we take its remainder when divided by 3. If the codon value is 10, rule (1) will be selected, since $10 \equiv 1 \pmod 3$.

In this problem, each individual represents an algebraic expression. The grammar is used to map the genotype (a sequence of integers) to phenotype (an algebraic expression).

A simple grammar was used (this is an example of BNF):

$$
\begin{aligned}
\langle \textbf{expr} \rangle ::= \ & \langle \text{expr} \rangle \ \langle \text{op} \rangle \ \langle \text{expr} \rangle && (0)\\
| \ & (\langle \text{expr} \rangle \ \langle \text{op} \rangle \ \langle \text{expr} \rangle) && (1)\\
| \ & \langle \text{expr} \rangle \ / \ \langle \text{const} \rangle && (2)\\
| \ & (\langle \text{expr} \rangle \ / \ \langle \text{const} \rangle) && (3)\\
| \ & \langle \text{var} \rangle && (4)
\end{aligned}
$$

$$
\begin{aligned}
\langle \text{op} \rangle ::= \ & + && (0)\\
| \ & - && (1)\\
| \ & * && (2)
\end{aligned}
$$

$$
\begin{aligned}
\langle \text{var} \rangle ::= \ & x && (0)\\
| \ & \langle \text{const} \rangle && (1)
\end{aligned}
$$

$$
\begin{aligned}
\langle \text{const} \rangle ::= \ & \langle \text{const} \rangle \ \langle \text{op} \rangle \ \langle \text{const} \rangle && (0)\\
| \ & (\langle \text{const} \rangle \ \langle \text{op} \rangle \ \langle \text{const} \rangle) && (1)\\
| \ & \langle \text{const} \rangle \ / \ \langle \text{const} \rangle && (2)\\
| \ & (\langle \text{const} \rangle \ / \ \langle \text{const} \rangle) && (3)\\
| \ & 1.0 && (4)\\
| \ & 3.0 && (5)
\end{aligned}
$$

The first (bolded) symbol is the start symbol, <expr>. Nonterminal symbols are enclosed in angular brackets ("<" and ">"). Codons are used one at a time. The first codon determines how to replace the start symbol <expr>. Each next codon value is used for the left-most nonterminal symbol in the so-far-obtained expression.

Genotype-to-phenotype mapping example

Each individual's chromosome is a sequence of codons (integers). Example individual is $(120, 44, 42, 96, 189, 64, \ldots)$. We explain how does this sequence map to an expression. The start symbol is <expr>, the first symbol in the grammar. There are five rules given which can replace <expr>. First codon determines which rule is applied. First codon value is 120, so rule number 0 is applied, because $120 \equiv 0 \pmod 5$. In this case, <expr> is replaced by <expr><op><expr>. We denote this as <expr> → <expr><op><expr>. Now, second codon determines which rule will be applied to <expr>, the first of the three symbols we obtained. Second codon value is 44, so the rule number 4 is applied ($44 \equiv 4 \pmod 5$), i. e., <expr><op><expr> → <var><op><expr>. Third codon

value, 42, determines that <var> will be replaced by x (grammar gives just 2 rules for <var> and $42 \equiv 0 \pmod 2$), so we use the rule number 0), i. e., <var><op><expr> → x<op><expr>. Now, x is a terminal symbol (grammar does not provide any rules to replace it), so the next symbol we concentrate on is <op>. Next codon $96 \equiv 0 \pmod 3$), so x<op><expr> → x+<expr>. The next two codons (189 and 64) result in $x + \langle \text{expr} \rangle \rightarrow x + \langle \text{var} \rangle \rightarrow x + x$.

If there are any remaining codons, they are simply left unused. On the other hand, if all the codons are used and there are still nonterminal symbols left in the expression, the individual is considered "invalid" (so that the selection process removes it).

Each time a particular individual's genotype is mapped to its phenotype, the same output is generated. This happens because the same choices are made each time.
This mapping is not injective – due to the fact that codons are considered modulo the number of possibilities, different codons may result in the selection of a same production rule. For more details on GE, one can check the first publication [O'Neill and Ryan 2001].

2.4 Evolutionary algorithm

Evolutionary algorithms are probabilistic algorithms that mimic the process of evolution, especially of natural selection. The population of candidate solutions, called *individuals*, is evolved towards better solutions. The initial population is usually chosen randomly. Each next generation is selected through a combination of genetic operators: crossover, mutation and duplication. The crossover is used to create new individuals from existing ones. The mutation, if it happens, alters the new individual by changing part of its genes. The duplication, which is rarely used, also alters the new individual, by copying the part of its genes at the end of the genotype. Not all individuals have the same chance of being selected for creating the next generation. All individuals are evaluated by a fitness function, which measures how well does this individual suit the problem. Better individuals have higher chances of being selected for crossover. We focus now on the details of the algorithm in this study.

Evaluation, initialization, crossover and mutation of individuals

The aim is to find an individual that will, using this grammar, map to the expression describing the given data. The population has 500 individuals. Individuals initially consist of 30 codons – each codon contains 8 bits, which are chosen randomly; so each codon is a random integer between 0 and 255 ($2^8 - 1$). A given sample of Frobenius numbers is used for the evaluation of individuals: fitness for this problem is given by the sum, taken over 40 function values, of the errors between the evolved and target function (Frobenius number).

The search is done by generational evolutionary algorithm. As the individuals are simple bit-vectors, we do not have to employ any specific crossover or mutation operators. We used one-point crossover, in which parent bit-vectors are broken at some randomly chosen point, and the pieces are combined to generate offspring that inherit one piece from each parent. Parents were selected in a tournament manner: two best individuals out of five randomly chosen. Crossover probability was then 0.9. One-bit mutation was used, in which a randomly chosen bit is changed with probability of 0.1. Along these standard genetic operators, we use a codon duplication operator. A number of codons is randomly chosen, as is the starting position of the first codon. This subsequence of codons is then copied at the end of the genotype. Duplication probability for each individual is 0.1.

After 250 potential crossovers (each resulting with two new individuals), new generation consisted of the best 500 individuals among both parents and offspring. The search lasted for a 100 generations. It is not important to carefully optimize these parameters.

Result of the algorithm

The output of a program is created and interpreted in the following manner. After each iteration, the best individual found so far is printed – both its genotype and phenotype (the expression), as well as its fitness value. In case of ties, because the array of individuals is sorted by the individuals' fitness values, only one individual – the first one – is printed. The obtained expression is then simplified, respecting the rules of the programming language used (C/C++). For example, if there is a division operator between (positive) integer variables, it is interpreted as the floor of the actual division, since C discards the fractional part when dividing integers. Besides introducing the floor function, it is appropriate to remove the unnecessary parentheses and evaluate constants (e.g., $3 * (3 + (3 + 1))$ is replaced by 21). After every 10 generations, the errors between the evolved expression and the target function (and the sum of them, i.e. fitness value) are printed as well.

GE did not find a formula valid for all given quadruples, but it found one which had a very high fit, i. e., a very small error:

$$g(x, x+3, 2x+1, 3x+7) \approx x\left(\left\lfloor \frac{x}{3} \right\rfloor - \left\lfloor \frac{x}{21} \right\rfloor\right) - 1.$$

It was not hard to notice that this formula is valid in the case of $x = 3k + 1$ and $k \geqslant 5$, simply by looking at the error values (0 for such x).

Theorem 1 *Frobenius number of the quadruple* $(3k+1, 3k+4, 6k+3, 6k+9)$*, for all* $k \geqslant 5$*, is*

$$g(3k+1, 3k+4, 6k+3, 6k+9) = (3k+1)\left(k - \left\lfloor \frac{3k+1}{21} \right\rfloor\right) - 1.$$

2.5 Proof of the conjectured formula

Our proof follows the method developed in the article *Frobenius numbers by lattice point enumeration* [Einstein, Lichtblau, Strzebonski, and Wagon 2007] and concrete details were found with the help of the algorithm, i.e., the Mathematica package they developed. Our proof is very similar to their proof for quadratic sequences. This is the reason why the author provides only the sketch of the proof for the case of $k = 7a$, without explaining the lattice point enumeration method (for understanding the proof and terminology, sections 1. – 4., 6. & 17. of [Einstein et al. 2007] should suffice).

Table 2 is basically the proof of the formula when $k = 7a$ ($a \in \mathbb{N}$). Proposed *elbows* determine the domain whose volume is $3k + 1 = 21a + 1$ (these elbows determine a domain whose volume is a linear function of a, so it suffices to check two cases), while *protoelbows* below show that the corresponding elbows are not in the fundamental domain.

Corner weights	$126a^2 + 27a$	$105a + 16$	$105a + 10$
Corners	$(0, 0, 3a)$	$(1, 1, 1)$	$(1, 2, 0)$
Elbows	$(0, 0, 3a + 1)$	$(0, 1, 2)$ $(2, 1, 0)$	$(0, 3, 0)$ $(0, 2, 1)$
Protoelbows	$(-2, 0, 3a + 1)$	$(-5, 1, 2)$ $(2, 1, -1)$	$(-1, 3, 0)$ $(-3, 2, 1)$

Corner weights	$126a^2 + 27a - 1$	$126a^2 + 27a - 2$	$126a^2 + 27a - 3$
Corners	$(2, 0, 3a - 1)$	$(4, 0, 3a - 2)$	$(6, 0, 3a - 3)$
Elbows	$(1, 0, 3a)$	$(3, 0, 3a - 1)$	$(5, 0, 3a - 2)$ $(7, 0, 0)$
Protoelbows	$(1, -2, 3a)$	$(3, -1, 3a - 1)$	$(0, 0, 0)$ $(7, 0, -3)$

Table 2: Description of fundamental domain for $(21a + 1, 21a + 4, 42a + 3, 42a + 9)$.

The Frobenius corner here is $(0, 0, 3a)$, so Frobenius number is

$$3a \cdot (6k+9) - (3k+1) = 3a \cdot (42a+9) - 21a - 1 = 126a^2 + 6a - 1 = (21a+1)(7a-a) - 1,$$

which is exactly what the formula gives for $k = 7a$. Other cases modulo 7 are resolved analogusly.

2.6 Other conjectures

Simply by changing the input data, the system can generate other conjectures. So long as the whole tuple depends on one parameter, it is not necessary to change the grammar. However, it is useful to introduce bigger constants (the author introduced 10). One can try to generalize the formulas for arithmetic and geometric sequences. Probably the simplest generalization of these two is linear first-order recursive sequence, e.g., sequence defined by $a_{n+1} = 3a_n + 2$.

GE system resulted with some conjectures in these cases. For even numbers, quadruples are not relatively prime. For odd numbers x, the formula depends on the remainder modulo 4. The following two formulas are conjectured:

$$f(x = 4k + 1, y = 3x + 2, z = 3y + 2, w = 3z + 2) =$$
$$48k^2 + 16k - 1 - 3(4k + 1)\left(k + \left\lfloor \frac{2k}{13} \right\rfloor + \left\lfloor \frac{2k + 6}{13} \right\rfloor - \left\lfloor \frac{2k + 19}{26} \right\rfloor\right),$$

$$f(x = 4k + 3, y = 3x + 2, z = 3y + 2, w = 3z + 2) =$$
$$48k^2 + 64k + 19 - 3(4k + 3)\left(k + \left\lfloor \frac{2k + 1}{13} \right\rfloor + \left\lfloor \frac{2k + 7}{13} \right\rfloor - \left\lfloor \frac{k + 3}{13} \right\rfloor\right).$$

Even the size of the tuple is easily changed, e.g., the system found the following conjecture as well:

$$g(6k + 1, 6k + 4, 6k + 7, 12k + 3, 12k + 9, 12k + 15) = (6k + 1)\left(k - \left\lfloor \frac{k}{13} \right\rfloor\right) - 1 \quad \forall k \geqslant 5.$$

One can even change the problem (again, simply by changing the input data to examples for that problem), but that would take us out of the scope of this work.

3 Benefits and limitations

Grammatical evolution can generate solutions in an arbitrary language by a simple change of the grammar – which is usually one short input file. This advantage emerges from the generality of context-free grammars, hence GE has broad generativity: it can be applied not only to the search of a formula, but also to anything that context-free grammars can describe.

Knowledge, embedded in a grammar's rules, speeds up the search process. In the present study, knowledge was basically the way expressions are created. Minor expectation of desired formula, the fact that the division operator should appear only with constant as divisor, was also embedded in the grammar. Introducing this expectation in the grammar significantly sped up the search: the formula was found more often (and after less generations) than when allowing divisor to contain a variable. The challenge in similar studies could be in embedding more complex expectations (e.g., recent results) in grammars. Constructing the grammar might require more creativity when one applies this method to different types of problems (say, constructing a grammar to find a graph with some particular property probably will not be such an easy task). However, given that grammars are the reason of great generalizability, the author believes that this advantage outweighs disadvantages.

Formulas found here could have been found using a more standard type of regression or using polynomial interpolation. However, because of the floor function, the formula proven here would be broken into multiple polynomial formulas. Far more important, these methods require the researcher to know or guess the model prior to applying them, and they are not so easy to generalize. On the other hand, fine tuning, i.e., getting the constants right, is a potential, but minor problem for GE. Formulas like the one proven here, ending with -1, can be missed by that -1 (in many runs of the program). Unlike a computer, a researcher can easily note and fix this type of problem.

Acknowledgments

The author would like to thank Armando Chapin Rodríguez, Professor Andrej Dujella, Jerry Marlow, and the anonymous referee whose suggestions helped improve this manuscript. The author was supported by the Croatian Science Foundation under the project no. 6422.

References

[Chomsky 1956] N. Chomsky. "Three models for the description of language." *IRE Transactions on Information Theory* 2 (1956), 113–124.

[Colton 2002] S. Colton. "The HR Program for Theorem Generation." In *Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002*, A. Voronkov (ed.), 285–289, 2002.

[Curtis 1990] F. Curtis. "On formulas for the Frobenius number of a numerical semigroup." *Mathematica Scandinavica* 67 (1990), 190–192.

[Dulmage and Mendelsohn 1964] A. L. Dulmage and N. S. Mendelsohn. "Gaps in the exponent set of primitive matrices." *Illinois J. Math.* 8(4) 12 1964 642–656.

[Einstein et al. 2007] D. Einstein, D. Lichtblau, A. Strzebonski, and S. Wagon. "Frobenius numbers by lattice point enumeration." *Integers: The Electronic Journal of Combinatorial Number Theory* 7(1) (2007), A15, 1–63.

[Erdős et al. 1995] P. Erdős, R. Faudree, T. J. Reid, R. Schelp, and W. Staton. "Degree sequence and independence in K(4)-free graphs." *Discrete Mathematics* 141(1–3) (1995), 285 – 290.

[Fajtlowicz and Waller 1986] S. Fajtlowicz and W. Waller. "On Two Conjectures of Graffiti." *Congressus Numerantium* 55 (1986), 51–56.

[Fenton et al. 2014] M. Fenton, C. McNally, J. Byrne, E. Hemberg, J. McDermott, and M. O'Neill. "Automatic innovative truss design using grammatical evolution." *Automation in Construction* 39 (2014), 59–69.

[Naur 1963] P. Naur. "Revised report on the algorithmic language Algol 60." *Communications of the ACM* 6(1) (1963), 1–17.

[O'Neill and Ryan 2001] M. O'Neill and C. Ryan. "Grammatical Evolution." *IEEE Transactions on Evolutionary Computation* 5(4) (2001), 349–358.

[O'Neill et al. 2002] M. O'Neill, A. Brabazon, and C. Ryan. "Forecasting market indices using evolutionary automatic programming: A case study." In *Genetic Algorithms and Genetic Programming in Economics and Finance*. S.-H. Chen (ed.), 174–195. Kluwer Academic Publishers, Dordrecht, 2002.

[Ramírez Alfonsín 2005] J. L. Ramírez Alfonsín. *The Diophantine Frobenius Problem*. Oxford Lecture Series in Mathematics and Its Applications. OUP Oxford, 2005.

[Sylvester 1882] J. J. Sylvester. "On Subvariants, i.e. Semi-Invariants to Binary Quantics of an Unlimited Order." *American Journal of Mathematics* 5(1) (1882), pp. 79–136.