Both questions are undecidable for *some* instances of equation (2). For *each* equation (2) the information contained in the sequence of k answers to the yes/no question "does the equation $D(n_0, x_1, x_2, \ldots, x_m) = 0$ have a solution for $n = 1, 2, \ldots, k$?" contains only $\log k$ bits of information (knowing how many equations have solutions is enough to determine exactly which equations have solutions). This information can be substantially compressed. However, for some instances of equation (2) the information contained in the sequence of k answers to the yes/no question "does the equation $D(n_0, x_1, x_2, \ldots, x_m) = 0$ have infinitely many solutions for $n = 1, 2, \ldots, k$?" contains about k bits of information, that is, the information cannot be algorithmically compressed. For more, see [12].

2.5 The Four Color Theorem

The four color theorem, first conjectured in 1853 by Francis Guthrie, states that every plane separated into regions may be colored using no more than four colors in such a way that no two adjacent regions receive the same color. Two regions are called *adjacent* if they share a border segment, not just a point. Regions must be contiguous, that is, the plan has no exclaves.

In graph-theoretical terms, the four color theorem states that the vertices of every planar graph can be colored with at most four colors so that no two adjacent vertices receive the same color. Shortly, every planar graph is four-colorable.

The four color theorem was proved in 1977 [13, 14] (see also [15]) using a computer-assisted proof that consists of constructing a finite set of "configurations", and then proving that each of them is "reducible"—which implies that no configuration with this property can appear in a minimal counter-example to the theorem. Checking the correctness of the original proof is a very difficult task. It implies, among other things, checking the descriptions of 1476 graphs, checking the correctness of the programs, proving the correctness of the compiler used to compile the programs, and checking the degree of reliability of the hardware used to run the programs. This computer-assisted proof generated lots of mathematical and philosophical discussions around the notion of acceptable mathematical proof; see, for example, [16-18]. Various partial independent verifications have been obtained, though it appears that there is no verification in its entirety. The formal confirmation announced in [19] uses the equational logic program Coq (see [20] for a recent presentation of the formal proof). The following quote from the concluding discussion in [19] is relevant for the current status of the proof (emphasis added):

https://doi.org/10.25088/ComplexSystems.18.3.267

> However, an argument can be made that our "proof" is not a proof in the traditional sense, because it contains steps that can never be verified by humans. In particular, we have not proved the correctness of the compiler we compiled our programs on, nor have we proved the infallibility of the hardware we ran our programs on. These have to be taken on faith, and are conceivably a source of error.... Apart from this hypothetical possibility of a computer consistently giving an incorrect answer, the rest of our proof can be verified in the same way as traditional mathematical proofs. We concede, however, that *verifying a computer program is much more difficult than checking a mathematical proof of the same length*.

A program $\Pi_{\text{fourcolortheorem}}$ that systematically generates all planar graphs and checks if each one is colorable with four colors and stops when the first counter-example is found will never halt if and only if the theorem is true. However, this program will be quite long because testing the planarity of a graph is difficult. A better solution is to use the Diophantine representation of the four color theorem proposed in [9]:

$$F(n, t, a, \ldots) = 0. \tag{3}$$

Equation (3) has no solution if and only if every planar graph can be colored with at most four colors so that no two adjacent vertices receive the same color. Based on equation (3), we can write the program Π_F as in Section 2.4, which can be taken as $\Pi_{\text{fourcolortheorem}}$.

Actually, it is better to use a pre-Diophantine representation given by the following conditions. Without restricting the generality, we consider the maps T_n consisting of the points (x, y) such that $J(x, y) \le Q = \left(n^2 + 3n\right)/2$, where J is Cantor's bijection $J(x, y) = \left((x+y)^2 + 3x + y\right)/2$. Given a four-coloring of T_n, $t_0, t_1, \ldots, t_Q$ there exist (and can be effectively computed) s, t such that for all $0 \le i \le Q$ (the integer remainder function is denoted by rem):

$$t_i = \text{rem}(t, 1 + s(i+1)).$$

In other words, the sequence $t_0, t_1, \ldots, t_Q$ can be coded by s and t.

Every sequence $u_0, u_1, \ldots, u_Q$ with $u_i < 4$ can be represented by some $u \le R = (1 + 4(Q+2)!)^{Q+1}$ such that

$$u_i = \text{rem}(u, 1 + 4(Q+2)!\,(i+1)).$$

Finally, there is a map (say, T_n) that cannot be colored in four colors if and only if the following condition is satisfied:

$$(\exists\, n, t, s)\,(\forall\, u \le R)\,(\exists\, x, y)\,(x + y \le n)\,[A \vee B],$$

https://doi.org/10.25088/ComplexSystems.18.3.267

where

$$A = u_{J(x,y)} \geq 4,$$

$$\begin{aligned} B = \big[&\big(t_{J(x,y)} = t_{J(x+1,y)} \wedge u_{J(x,y)} \neq u_{J(x+1,y)}\big) \vee \\ &\big(t_{J(x,y)} \neq t_{J(x+1,y)} \wedge u_{J(x,y)} = u_{J(x+1,y)}\big) \vee \\ &\big(t_{J(x,y)} = t_{J(x,y+1)} \wedge u_{J(x,y)} \neq u_{J(x,y+1)}\big) \vee \\ &\big(t_{J(x,y)} \neq t_{J(x,y+1)} \wedge u_{J(x,y)} = u_{J(x,y+1)}\big)\big]. \end{aligned}$$

A simple inspection shows that the given condition is computable, so the four color theorem is of the form $(\forall\, n)\, P(n)$, where P is a computable predicate.

2.6 The Riemann Hypothesis

The Riemann hypothesis is probably the most famous and important conjecture in mathematics. It appears in Hilbert's eighth problem [5]: the nontrivial complex zeros of Riemann's zeta function, which is defined for $\mathrm{Re}(s) > 1$ by

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s},$$

lie exactly on the line $\mathrm{Re}(s) = 1/2$.

According to Matiyasevich [8, pp. 119-121], the negation of the Riemann hypothesis is equivalent to the existence of positive integers k, l, m, n satisfying the following six conditions (here $x \mid z$ means "x divides z"):

1. $n \geq 600$,
2. $\forall\, y < n\big[(y+1) \mid m\big]$,
3. $m > 0\ \&\ \forall\, y < m\big[y = m \vee \exists\, x < n\big[\neg\ \big[(x+1) \mid y\big]\big]\big]$,
4. $\mathrm{explog}(m-1, l)$,
5. $\mathrm{explog}(n-1, k)$,
6. $(l-n)^2 > 4\, n^2\, k^4$,

and $\mathrm{explog}(a, b)$ denotes the predicate

$$\exists\, x \left[x > b+1\ \&\ \left(1 + \frac{1}{x}\right)^{x\,b} \leq a+1 < 4\left(1 + \frac{1}{x}\right)^{x\,b} \right].$$

An inspection of these conditions shows that the Riemann hypothesis is of the form $\forall\, n$, $R(n)$, where R is a computable predicate. Hence,

https://doi.org/10.25088/ComplexSystems.18.3.267

one can write a program Π_{Riemann} such that the Riemann hypothesis is false if and only if Π_{Riemann} halts.

2.7 The Collatz and Palindrome Conjectures

When he was a student, L. Collatz posed the following problem: given any integer seed a_1, there exists a natural N such that $a_N = 1$, where

$$a_{n+1} = \begin{cases} \dfrac{a_n}{2}, & \text{if } a_n \text{ is even,} \\ 3\,a_n + 1, & \text{otherwise.} \end{cases}$$

This is known as Collatz's conjecture, the Syracuse conjecture, the $3\,x + 1$ problem, Kakutani's problem, Hasse algorithm, or Ulam's problem. There is a huge amount of literature on this problem and various natural generalizations: see [21-23]. Erdös has said (cf. [21]) that "Mathematics may not be ready for such problems."

Does there exist a program Π_{Collatz} such that Collatz's conjecture is false if and only if Π_{Collatz} halts?

First, we note that a brute-force tester, that is, a program that will enumerate all seeds and try to find an iteration equal to 1 for each of them, may never stop in two different cases: (*a*) because the conjecture is indeed true, or (*b*) because for some specific seed a_1 there is no N such that $a_N = 1$. It is not clear how to differentiate these cases; even worse, it is not clear how to refute (*b*) using a brute-force tester.

A simple nonconstructive argument answering our question in the affirmative appears in [3]. Indeed, observe first that the set

$$\mathit{Collatz} = \{a_1 : a_N = 1, \text{ for some } N \geq 1\}$$

is computably enumerable. Collatz's conjecture requires proving that *Collatz* does indeed contain all positive integers.

If *Collatz* is not computable, then the conjecture is false, and any program that eventually halts can be taken as Π_{Collatz} as (*a*) is ruled out. If *Collatz* is computable, then we can write a program Π_{Collatz} to find an integer not in *Collatz*: the conjecture is true if and only if Π_{Collatz} never stops.

Now we present the palindrome conjecture. The reverse (mirror) of a number is the number formed with the same decimal digits but written in the opposite order. For example, the mirror of 12 is 21, the mirror of 131072 is 270131, and so on. Start with the decimal representation of a natural *a*, reverse the digits, and add the constructed number to *a*. Iterate this process until the result is a palindrome. Following [24], the palindrome conjecture states that for every natural *a*, a palindrome number will be obtained after finitely many iterations of the given procedure.

https://doi.org/10.25088/ComplexSystems.18.3.267

The same nonconstructive argument used for Collatz's conjecture applies to the palindrome conjecture: there exists a program $\Pi_{\text{palindrome}}$ such that the palindrome conjecture is true if and only if $\Pi_{\text{palindrome}}$ never stops.

Collatz and palindrome conjectures have the following general form. Let $a \in \mathbf{N}$ and let T be a computable function from naturals to naturals. The conjecture associated to (a, T) is: for each $x \in \mathbf{N}$, $T^i(x) = a$, for some $i > 0$.

Considering the set

$$B(a, T) = \left\{x \in \mathbf{N} : T^i(x) = a, \text{ for some } i > 0\right\}, \tag{4}$$

the conjecture associated to (a, T) becomes equivalent with the equality $B(a, T) = \mathbf{N}$. The argument used for Collatz's conjecture applies to this general case too, so one can prove in a nonconstructive way the existence of a program $\Pi_{(a,T)}$ such that the conjecture associated to (a, T) is true if and only if $\Pi_{(a,T)}$ never stops.

3. The Halting Problem

In Section 2 the halting property of various programs repeatedly appeared. It is time to ask the question: can the halting problem be solved by a program? As all of our programs have a very specific form—they have no input and each of them either stops (in which case the output is a natural number) or never stops—we show with a simple argument that the halting problem, that is, the problem of whether or not such a program eventually stops, is unsolvable by any program. This means that there is no program **H** with the following three properties.

1. **H** accepts as input any program of the given type.
2. **H** eventually halts.
3. **H** produces the output 1 if the input program eventually stops, or 0 in case the input program never stops.

Here is an information-theoretic analysis of the existence of the hypothetical program **H**. Assume that there exists a halting program **H** with the three properties. Using **H** we construct the following (legitimate) program **P**.

1. Read a natural N.
2. Generate all programs up to N bits in size.
3. Use **H** to check each generated program for whether or not it halts.

https://doi.org/10.25088/ComplexSystems.18.3.267

4. Simulate the running of the remaining programs.

5. Output 1 plus the biggest value output by these programs.

The program **P** halts for every natural N. Indeed, the number of programs less than N bits in size is finite, so by the assumption that **H** can decide the halting status of every program in a finite amount of time, we can filter out all nonhalting programs. The remaining programs (certainly, finitely many) can be run and after a finite (maybe very long) computation they will all halt and each will produce a natural number as output. Did we obtain a contradiction?

To answer the question we have to ask the right auxiliary question: how long is **P**? The answer is that **P** is about $\log_2 N$ bits. Indeed, we need about $\log_2 N$ bits to code N in binary, and the rest of the program **P** is a constant, say c. Hence, the length of **P** is $\log_2 N + c$ bits.

Now observe that there is a big difference between the size of **P** and the size of the output produced by **P**. Indeed, for large enough N, **P** belongs to the set of programs having less than N bits because $\log_2 N + O(1) < N$. Hence, in this case, **P** generates itself at some stage of the computation. As **P** always halts, the program **H** decides that **P** stops, so **P** is run as a part of the simulation (inside the computation of **P**) and produces a natural number as a result. But the program **P** itself will output a natural number that is different from every output produced by a simulated computation, in particular from the output produced by **P** itself, which is a contradiction.

This proof (see [12] for more details) shows that in general there is no method, no uniform procedure, to test whether an arbitrary program eventually stops or not. Of course, this does not imply that for some infinite class of programs we cannot find a program to decide their halting status. Actually, there are infinite sets of programs for which the halting problem is decidable, for example, the class of primitive recursive programs (which are all total).

What is the situation with the programs associated to the problems discussed before? Can we hope to decide for each of them whether it halts or not? For some programs, such as Π_{Fermat}, we know the answer: the program never stops as certified by A. Wiles' proof of Fermat's last theorem. For the program Π_{Riemann} the answer is not known. We currently cannot even explicitly write the program Π_{Collatz}. (We conjecture that the statement "Π_{Collatz} never stops" is independent of the Zermelo-Fraenkel set theory with the axiom of choice, or ZFC.) For some programs Π the statement "Π halts" is independent of ZFC. Such a statement has to be true. A proof of independence is an alternative proof, admittedly not usual, of the truth of the statement.

https://doi.org/10.25088/ComplexSystems.18.3.267

4. A Computational Method for Evaluating the Complexity of Mathematical Problems

We now return to Fermat's last theorem and to the fact that it is equivalent to the statement "Π_{Fermat} never halts". We do not propose to prove Fermat's last theorem by showing that Π_{Fermat} never halts, but to use the program Π_{Fermat} as a way to measure the complexity of Fermat's last theorem.

We do this by counting the number of bits necessary to specify Π_{Fermat} in some fixed "universal formalism" (e.g., a universal self-delimiting Turing machine [12]). Of course, there are many programs equivalent to Π_{Fermat}, so a natural way to evaluate their complexity is to consider the smallest such program.

The choice of the universal formalism used to code programs is irrelevant up to an additive constant, so if a problem is significantly more complex in some fixed formalism than in another one, then it will continue to be more complex in any other formalism. However, the proposed measure is uncomputable [12], so we have to work with an upper bound on the size of a program "describing" the conjecture, problem, or theorem.

In practice, to evaluate the complexity of a problem P we need to effectively obtain the program Π_P and compute its size in bits. This gives an upper bound on the complexity of Π_P, and hence on the difficulty of P. But, as we have seen with the Collatz and palindrome conjectures, even this type of approximation may not be achievable in all cases. We cannot evaluate a bound on the difficulty of a problem P if we do not know at least one "explicit" program Π_P.

5. Finitely Refutable Problems

It is time to ask the question: What is the class of problems whose complexity can be evaluated with the method proposed in Section 4? We will not answer this question, which is open, but do give an example of a large class of problems to which the method applies. With Pythagoras' dictum "all is number" as a guiding principle we will look at finite numerical tests.

Let $\mathbf{N}$ denote the set of positive integers and for every $k \in \mathbf{N}$ consider a predicate P on $\mathbf{N}$.

Consider the formula

$$f = Q_1\, n_1\, Q_2\, n_2 \ldots Q_k\, n_k\, P(n_1, n_2, \ldots, n_k)$$

where $Q_1, Q_2, \ldots, Q_k \in \{\forall, \exists\}$ are quantifier symbols. In analogy with the arithmetic classes, we say that f is in the class $\hat{\Pi}_s$ or $\hat{\Sigma}_s$ if the quantifier prefix of f starts with $\forall$ or $\exists$, respectively, and contains

https://doi.org/10.25088/ComplexSystems.18.3.267

$s-1$ alternations of quantifier symbols. When P is computable, then f is in Π_s or Σ_s, respectively. It is sufficient to consider only such formulas f in which no two consecutive quantifier symbols are the same. In the remainder of this section we make this assumption without special mention. With f as given, we have $s = k$.

As usual, with P as given, we write $P(n_1, \ldots, n_k)$ instead of $P(n_1, \ldots, n_k) = 1$ when $n_1, \ldots, n_k$ are elements of $\mathbf{N}$. Thus, $\neg P(n_1, \ldots, n_k)$ if and only if $P(n_1, \ldots, n_k) = 0$. Moreover, since we consider variable symbols only in the domain $\mathbf{N}$, if f is any formula in first-order logic, we write f is true instead of f is true in $\mathbf{N}$.

Let Γ_s be one of the classes $\hat{\Pi}_s$, $\hat{\Sigma}_s$, Π_s, or Σ_s. We refer to the task of proving or refuting a first-order logic formula as a *problem* and, in particular, to problems expressed by formulas in Γ_s as *Γ_s-problems*.

We say that a problem is being *solved* if the corresponding formula is proved or disproved to be true, that is, if the truth value of the formula is determined. A problem is said to be *finitely solvable* if it can be solved by examining finitely many cases.

For example, consider the predicate

$$P(n) = \begin{cases} 1, & \text{if } n \text{ is even or } n = 1 \text{ or } n \text{ is a prime,} \\ 0, & \text{otherwise,} \end{cases}$$

that is, $P(n) = 0$ if and only if n is an odd number greater than 1 which is not a prime. Then the conjecture expressed by the formula $(\forall\, n)\, P(n)$ is finitely solvable; indeed, it is sufficient to check all n up to 10 to refute this conjecture.

Goldbach's conjecture is a Π_1-problem. To express it let $P_{\text{Goldbach}} : \mathbf{N} \to \{0, 1\}$ be such that

$$P_{\text{Goldbach}}(n) = \begin{cases} 1, & \text{if } n \text{ is odd or } (n \text{ is even and } n \text{ is the sum of two primes}), \\ 0, & \text{otherwise.} \end{cases}$$

Thus, $f_{\text{Goldbach}} = (\forall\, n)\, P_{\text{Goldbach}}(n)$ is true if and only if Goldbach's conjecture is true.

Similarly, the Riemann hypothesis is a Π_1-problem. By a result given in [9], the Riemann hypothesis can be expressed in terms of the function $\delta_{\text{Riemann}} : \mathbf{N} \to \mathbf{R}$ defined by

$$\delta_{\text{Riemann}}(k) = \prod_{n<k} \prod_{j \le n} \eta_{\text{Riemann}}(j),$$

https://doi.org/10.25088/ComplexSystems.18.3.267

where

$$\eta_{\text{Riemann}}(j) = \begin{cases} p, & \text{if } j = p^r \text{ for some prime } p \text{ and some } r \in \mathbf{N}, \\ 1, & \text{otherwise.} \end{cases}$$

The Riemann hypothesis is equivalent with the assertion that for all $n \in \mathbf{N}$

$$\left(\sum_{k \le \delta_{\text{Riemann}}(n)} \frac{1}{k} - \frac{n^2}{2} \right)^2 < 36\, n^3.$$

If we set

$$P_{\text{Riemann}}(n) = \begin{cases} 1, & \text{if } \left(\sum_{k \le \delta_{\text{Riemann}}(n)} \frac{1}{k} - \frac{n^2}{2} \right)^2 < 36\, n^3, \\ 0, & \text{otherwise.} \end{cases}$$

Then, $f_{\text{Riemann}} = (\forall\, n)\, P_{\text{Riemann}}(n)$ is true if and only if the Riemann hypothesis is true. Clearly, P_{Riemann} is decidable, therefore, the Riemann hypothesis is a Π_1-problem.

What is the "commonality" of all problems in classes $\hat{\Pi}_s$ and $\hat{\Sigma}_s$?

For $s \in \mathbf{N}$, let $\hat{\Gamma}_s$ denote any of $\hat{\Pi}_s$ and $\hat{\Sigma}_s$, and let Γ_s denote any of Π_s and Σ_s. Let

$$f = Q_1\, n_1\, Q_2\, n_2 \ldots Q_s\, n_s\, P(n_1, n_2, \ldots, n_s)$$

with $s \in \mathbf{N}$, where $Q_1, Q_2, \ldots, Q_s$ are alternating quantifier symbols.

Following [25], we define a test set for f to be a set $T \subseteq \mathbf{N}^s$ such that f is true in $\mathbf{N}^s$ if and only if it is true in T. The problem f is finitely solvable if there is a finite test set for f. In [25] the following result was proved:

Every $f \in \hat{\Gamma}_s$ is finitely solvable.

In other words, a solution to each mass problem (i.e., a problem having an infinite number of instances or cases) in the given classes can be obtained by inspecting only finitely many instances of the problem. As we might expect, this fact cannot be used to obtain a uniform algorithmic way of solving these types of problems because the finite test set cannot be computed even for all problems in the class Π_1: There is no constructive proof showing that every $f \in \Pi_1$ has a finite test set.

https://doi.org/10.25088/ComplexSystems.18.3.267

6. The Power and the Limits of the Method

Our analysis gives a new method of comparing the difficulties of two or more finitely refutable problems. The main obstacle is the noncomputability of the measure [12]. However, by working with upper bounds, we can obtain a practical method for evaluating the complexity that allows a relative ranking of problems.

In weighting the importance of computing the *exact value* of the complexity measure, recall Knuth [26]: "premature optimization is the root of all evil" and Rabin [27] "we should give up the attempt to derive results and answers with complete certainty."

The method can be applied to every Π_1-problem. All problems discussed in Section 2 can be analyzed with this method [3]. Trying to reduce the length of a program is in general possible; of course, proving minimality is, in general, impossible [12].

The method proposed is certainly not universal. Let us discuss here the class of Π_1-problems. Not every mathematical statement is a Π_1-problem. For instance, the twin primes conjecture discussed in Section 2.2 is not a Π_1-problem. Writing

$$P_{\mathrm{TP}}(n, m) = \begin{cases} 1, & m > n \text{ and } m \text{ and } m+2 \text{ are primes,} \\ 0, & \text{otherwise,} \end{cases}$$

this conjecture can be stated as

$$f_{\mathrm{TP}} = \forall\, n \exists\, m\, P_{\mathrm{TP}}(n, m).$$

The formula f_{TP} is in the class Π_2. Bennett conjectured in [28] that most mathematical conjectures can be settled indirectly by proving stronger conjectures. For the twin primes conjecture a stronger Π_1-problem is obtained as follows. Consider the predicate

$$P'_T(n) = \begin{cases} 1, & \text{if there is } m \text{ with } 10^{n-1} \le m \le 10^n,\ m \text{ and } m+2 \text{ primes,} \\ 0, & \text{otherwise.} \end{cases}$$

Let $f'_T = (\forall\, n)\, P'_T(n)$. Thus, f'_T gives rise to a Π_1-problem and, if f'_T is true, then f_T is also true (but the converse is not necessarily true).

However, there exists a program Π_{TP} such that the twin primes conjecture is true if and only if Π_{TP} never halts. As in Collatz's case, the argument is nonconstructive and based on the fact that the set

$$\mathrm{TP} = \{n : \forall\, n \exists\, m > n \text{ such that } m \text{ and } m+2 \text{ are primes}\}$$

is computably enumerable.

The method depends on the chosen universal Turing machine, that is, our working framework. Changing the universal machine will result in changes of the complexity value, but not in relative compari-

https://doi.org/10.25088/ComplexSystems.18.3.267

son between problems. The choice of the machine is irrelevant up to an additive constant, so if a problem is significantly more complex than another one with respect to a fixed universal machine, then it will continue to be more complex for any other machine. The method was used by relativizing to the halting problem. The same method can be used by relativizing to other unsolvable problems, for example, the totality problem. Going from the halting problem to the totality problem will increase the power of expression. For example, to the conjecture associated to (a, T) (see equation (4)), we can associate the program $\Gamma_{a,T}(x)$ defined by

$$\Gamma_{a,T}(x) = \min_i \left[T^i(x) = a\right].$$

The conjecture associated to (a, T) is true if and only if $\Gamma_{a,T}(x)$ is total. Writing the program for $\Gamma_{a,T}(x)$ is simple, but in the special cases of the Collatz, palindrome, and twin primes conjectures writing the corresponding $\Pi_{a,T}$ program is problematic (we are only able to prove its existence).

In [29] Kim discusses the possibility that the Poincaré conjecture (a recently solved problem in topology, see [30-32]) is equivalent to the unsolvability of a Diophantine equation (see also [33]). If this is true, then our method would also offer, at least in principle, an indication of the difficulty of the Poincaré conjecture.

7. Conclusions

We have presented a computational method for evaluating the complexity of mathematical problems. The method, inspired by [1], is based on the possibility of completely describing complex mathematical problems, such as the Riemann hypothesis, in terms of very simple programs.

If a mathematical problem, irrespective of its nature, can be equivalently expressed in terms of the property that a certain associated program eventually halts, then the proposed method applies. For example, the method applies to every Π_1-problem. Specific instances of such problems are, for example, Fermat's last theorem, the Goldbach conjecture, the four color problem, the Riemann hypothesis, Hilbert's tenth problem, the Collatz problem, the palindrome conjecture, and the twin primes conjecture. As an illustration, according to this complexity measure, the Riemann hypothesis is about twice as difficult as the Goldbach conjecture [3]. Although the method applies to both the Collatz and twin primes conjectures, it is an open question whether one can effectively evaluate the complexity of these problems.

Our method, which is a refinement below the first Turing degree, provides a total order in the class of finitely refutable problems. The

https://doi.org/10.25088/ComplexSystems.18.3.267

difficulty of the problem is additive (modulo the constants involved due to the choice of the universal Turing machine). The same method can be used by relativizing to other unsolvable problems different from the halting problem (e.g., the totality problem).

The scalability of the measure, both in terms of ordering, the role of the additive constants involved, and its relativization to various unsolvable problems are open questions.

In Part 2 of this study we will present a formalism for uniformly evaluating the complexity of the problems discussed in this paper and a ranking of those problems will be presented.

Acknowledgment

We thank Professor John Casti for discussions and comments regarding the topic of this paper which improved our presentation. We also thank the anonymous referee for very helpful comments and suggestions.

This work was supported in part by the Andrea von Braun Foundation, Munich, under the grant for "Artistic Forms and Complexity".

References

[1] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc., 2002.

[2] "2010 Mathematics Subject Classification (MSC2010)." (Sep 17, 2009) www.ams.org/msc.

[3] C. S. Calude, E. Calude, and M. J. Dinneen, "A New Measure of the Difficulty of Problems," *Journal for Multiple-Valued Logic and Soft Computing*, **12**(3-4), 2006 pp. 285-307.

[4] T. Lampert, "Wittgenstein on the Infinity of Primes," *History and Philosophy of Logic*, **29**(1), 2008 pp. 63-81. doi.10.1080/01445340701507569.

[5] D. Hilbert, "Mathematical Problems," *Bulletin of the American Mathematical Society,* **8**(10), 1902 pp. 437-479.

[6] T. Oliveira e Silva. "Goldbach Conjecture Verification." (Jul 28, 2009) www.ieeta.pt/~tos/goldbach.html.

[7] Y. V. Matiyasevich. "Hilbert's Tenth Problem. " (Nov 30, 2008) logic.pdmi.ras.ru/Hilbert10/stat/stat-eng.htm.

[8] Y. V. Matiyasevich, *Hilbert's Tenth Problem*, Cambridge, MA: The MIT Press, 1993.

[9] M. Davis, Y. V. Matiyasevich, and J. Robinson, "Hilbert's Tenth Problem. Diophantine Equations: Positive Aspects of a Negative Solution," *Mathematical Developments Arising from Hilbert Problems* (F. E. Browder, ed.), Providence, RI: American Mathematical Society, 1976 pp. 323-378.

https://doi.org/10.25088/ComplexSystems.18.3.267

[10] E. W. Weisstein. "Sturm Function" from Wolfram *MathWorld*—A Wolfram Web Resource. mathworld.wolfram.com/SturmFunction.html.

[11] A. Tarski, *A Decision Method for Elementary Algebra and Geometry*, 2nd ed., Berkeley: University of California Press, 1951.

[12] C. S. Calude, *Information and Randomness: An Algorithmic Perspective*, 2nd ed. (revised and extended), Berlin: Springer-Verlag, 2002.

[13] K. Appel, W. Haken, and J. Koch, "Every Planar Map Is Four-Colorable, I: Discharging," *Illinois Journal of Mathematics*, **21**(3), 1977 pp. 429-490.

[14] K. Appel and W. Haken, "Every Planar Map Is Four-Colorable, II: Reducibility," *Illinois Journal of Mathematics,* **21**(3), 1977 pp. 491-567.

[15] R. Wilson, *Four Colours Suffice: How the Map Problem Was Solved*, London: Penguin, 2002.

[16] A. S. Calude, "The Journey of the Four Colour Theorem through Time," *The New Zealand Mathematics Magazine*, **38**(3), 2001 pp. 27-35.

[17] C. S. Calude, E. Calude, and S. Marcus, "Passages of Proof," *Bulletin of the European Association for Theoretical Computer Science*, **84**, 2004 pp. 167-188.

[18] C. S. Calude, E. Calude, and S. Marcus, "Proving and Programming," *Randomness and Complexity: from Leibniz to Chaitin* (C. S. Calude, ed.), Singapore: World Scientific, 2007 pp. 310-321.

[19] N. Robertson, D. P. Sanders, P. Seymour, and R. Thomas. "The Four Color Theorem." (Nov 30, 2008) www.math.gatech.edu/~thomas/FC/fourcolor.html.

[20] G. Gonthier, "Formal Proof—The Four Color Theorem," *Notices of the American Mathematical Society*, **55**(11), 2008 pp. 1382-1393.

[21] J. C. Lagarias, "The $3x + 1$ Problem and Its Generalizations," *American Mathematical Monthly*, **92**, 1985 pp. 3-23.

[22] R. K. Guy, "Problem E16," *Unsolved Problems in Number Theory*, 3rd ed., (R. K. Guy, ed.), New York: Springer, 2004 pp. 330-336.

[23] J.-P. Davalan, "$3x + 1$, Collatz, Syracuse Problem." (Nov 30, 2008) pagesperso-orange.fr/jean-paul.davalan/liens/liens_syracuse.html.

[24] J.-P. Delahaye, "Déconcertantes Conjectures," *Pour la Science*, **367**, 2008 pp. 90-95.

[25] C. S. Calude, H. Jürgensen, and S. Legg, "Solving Finitely Refutable Mathematical Problems," *Finite versus Infinite. Contributions to an Eternal Dilemma* (C. S. Calude and G. Păun, eds.), London: Springer-Verlag, 2000 pp. 39-52.

[26] D. E. Knuth, "Structured Programming with Go To Statements," *ACM Computing Surveys (CSUR)*, **6**(4), 1974 pp. 261-301. doi.acm.org/10.1145/356635.356640.

[27] M. O. Rabin, "The Possibilities of Chance," *Out of their Minds: The Lives and Discoveries of 15 Great Computer Scientists* (D. Sasha and C. Lazare, eds.), New York: Copernicus, 1995 pp. 68-89.

[28] C. H. Bennett, "Chaitin's Omega," *Fractal Music, Hypercards, and More —: Mathematical Recreations from Scientific American Magazine* (M. Gardner, ed.), New York: W. H. Freeman, 1992 pp. 307-319.

https://doi.org/10.25088/ComplexSystems.18.3.267

[29] M. Kim."Why Everyone Should Know Number Theory." (Nov 30, 2008) www.ucl.ac.uk/~ucahmki/numbers.pdf.

[30] B. Kleiner and J. Lott. "Notes and Commentary on Perelman's Ricci Flow Papers." (Nov 30, 2008) math.berkeley.edu/~lott/ricciflow/perelman.html.

[31] G. Perelman. "The Entropy Formula for the Ricci Flow and Its Geometric Application." (Nov 30, 2008) www.arxiv.org/abs/math.DG/0211159.

[32] G. Perelman, "Ricci Flow with Surgery on Three-Manifolds." (Nov 30, 2008) www.arxiv.org/abs/math.DG/0303109.

[33] J. F. Manning, "Algorithmic Detection and Description of Hyperbolic Structures on Closed 3-Manifolds with Solvable Word Problem," *Geometry & Topology*, **6**, 2002 pp. 1-26. doi .10.2140/gt.2002.6.1.

https://doi.org/10.25088/ComplexSystems.18.3.267