Scientific Research Publishing

Journal of Computer and Communications, 2023, 11, *-*
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

The Simplest Possible Fully Correct Solution of the Clay Millennium Problem about P vs. NP. A Simple Proof That P ≠ NP = EXPTIME

Konstantinos E. Kyritsis

School of Economics, University of Ioannina, Ioannina, Greece
Email: ckiritsi@uoi.gr

How to cite this paper: Author 1, Author 2 and Author 3 (2023) Paper Title. *Journal of Computer and Communications*, **11**, *-*.
https://doi.org/10.4236/***.2023.*****

Received: **** **, ***
Accepted: **** **, ***
Published: **** **, ***

Abstract

In the current paper, I present probably the simplest possible abstract formal proof that P ≠ NP, and NP = EXPTIME, in the context of the standard mathematical set theory of computational complexity and deterministic Turing machines. My previous publications about the solution of the P vs. NP with the same result NP = EXPTIME, to be fully correct and understandable need the Remark 4.1 and its proof of the current paper. The arguments of the current paper in order to prove NP = EXPTME are even simpler than in my previous publications. The strategy to solve the P vs. NP problem in the current paper (and in my previous publications) is by starting with an EXPTIME-complete language (problem) and proving that it has a re-formulation as an NP-class language, thus NP = EXPTIME. The main reason that the scientific community has missed such a simple proof, is because of two factors 1) It has been tried extensively but in vain to simplify the solutions of NP-complete problems from exponential time algorithms to polynomial time algorithms (which would be a good strategy only if P = NP) 2) It is believed that the complexity class NP is strictly a subclass to the complexity class EXPTIME (in spite the fact that any known solution to any of the NP-complete problems is not less than exponential). The simplicity of the current solution would have been missed if 2) was to be believed true. So far the majority of the relevant scientific community has considered this famous problem not yet solved. The present results definitely solve the 3rd Clay Millennium Problem about P versus NP in a simple, abstract and transparent way that the general scientific community, but also the experts of the area, can follow, understand and therefore become able to accept.

Keywords

3rd Clay Millennium Problem, EXPTIME-Complete Problems,

NP-Complexity, P-Complexity

1. Introduction

The P versus NP problem is generally considered not yet solved and by the more careful expert researchers as not yet been known if it has been solved or not. Many have claimed solutions from 2000 to 2016. G. J. Woeginger (see G. J. Woeginger [1] and [2] Wikipedia) compiled a list of 62 purported proofs of P = NP from 1986 to 2016, of which 50 were proofs of P ≠ NP, 2 were proofs the problem is unprovable, and one was a proof that it is undecidable I myself read some very few short solutions from this list which turned out to be incorrect. But I did not analyse the long ones (some more than 70 pages). I do not know any researcher who has gone through all the solutions in the list of G. J. Woeginger, to find which, if any, of the 60 solutions is correct. Although for some solutions in this list, it is easy to prove that are not correct, no one has ever published any proof that all of them are not correct. Probably this should be the task of the Clay Mathematical Institute which sponsored the formulation of this problem as one of the 7 Millennium problems, in other words, to hire a group of experts to do this task. Nevertheless, according to the rules about the millennium problems of the Clay Mathematical Institute, the Institute is waiting for the community of relevant experts and researchers to indicate by citations if there is a correct solution to the P vs. NP problem. Most of these 62 solutions are not in the main journals of complexity theory and the reason is that the most widely read such journals avoid refereeing any solution of the P vs. NP problem for obvious or non-obvious reasons, except perhaps if it is from a very well-known and celebrated professor in the field of complexity. Therefore, there is an obvious social barrier to publishing solutions to this problem in relevant Journals that are widely read. Strangely enough, the monetary award for the solution to this problem had two opposite effects. First an increased number of researchers from all areas trying to solve it, and second an avoidance of the main Journals in the specialization area, to consider solutions to this problem for refereeing which of course would exclude correct solutions also.

For those that tried to solve it in the direction P = NP, there is a common confusion and mistake, that has been pointed out by Yannakakis M [3]. Still, it is better to have published results than non-published, and then let a large number of readers try to find errors or flaws in the solutions if there are any.

So here comes the need for a more challenging problem: Not only to solve the P versus NP problem, but also solve it in a simple, elegant and short way, so that the researchers will know a decisive proof that they can understand and control that P ≠ NP or not, so short that anyone familiar with the area, would discover any flaw or error if it existed. I must say that I am not a dedicated researcher of computational complexity but an interdisciplinary researcher, and I have also solved the 4th Clay Millennium problem in fluid dynamics (see [7]).

Two solution of the famous P versus NP problem in the direction P ≠ NP, and NP = EXPTIME, have been published by me in [4] [5] [6] [8] Kyritsis K. and in this paper, we present a very shorter simplification of the solution. Nevertheless, the above previous publications about the solution of the P vs. NP with the same result NP = EXPTIME, **to be fully correct and understandable they need the Remark 4.1 and its short proof of the current paper.**

The strategy to solve the P vs. NP problem in the current paper (and in my previous publications) is by starting with an EXPTIME-complete language (problem) and proving that it has a re-formulation as an NP-class language with verifier relation and certificate, thus NP = EXPTIME.

The main reason that the scientific community has missed so far such a simple proof, is because of two factors.

1) It has been tried extensively but in vain to simplify the solutions of NP-complete problems from exponential time algorithms to polynomial time algorithms (which would be a good strategy only if P = NP).

2) It is believed that the complexity class NP is strictly a subclass to the complexity class EXPTIME (in spite of the fact that any known solution to any of the NP-complete problems is not less than exponential).

The simplicity of the current solution would have been missed if 2) was to be believed because the current solution is based on the strategy of STARTING with an EXPTIME-complete language (problem) and proving that it can be re-formulated as an NP-class language, thus NP = EXPTIME. **The present paper definitely solves the 3rd Clay Millennium Problem about P versus NP in a simple and transparent way that the general scientific community, but also the experts of the area, can follow, understand and therefore become able to accept**.

In the history of mathematics, it is known that difficult problems that have troubled a lot the mathematicians turned out to have different proofs one simple and one very complex. Such an example is if the general 5th-order polynomial equation can be solved with addition, subtraction, multiplication, division and extraction of radicals starting from the coefficients. The famous mathematician Niels Henrik Abel gave a very simple proof, of not more than 5 pages. On the other hand, the proof of the same, by the E. Galois theory, is a whole book of dozens of pages!

A famous mathematician once said that "*Once a proof is known to a mathematical problem*, *then immediately after it becomes trivial*!"

It is important to mention, a statement, that is usually attributed to the famous mathematician Yuri Manin, that "A correct proof in mathematics is considered a proof only if it has passed the social barrier of being accepted and understood by the scientific community and published in accepted Journals".

Passing the obstruction of the social barrier, *sometimes is more difficult than solving the mathematical problem itself*!

It is similar to the solution of the P versus NP problem in this paper.

We will utilize in our proofs, the ***key abstraction*** of the existence of an

EXPTIME complete language, (it is known that it exists) without specifying which one, which will simplify much of the arguments. Then we prove that there is a reformulation of it that fits the definition of a language being an NP-class language.

The P vs. NP is not a problem that a computer experiment can decide, but rather a problem that requires the correct arguments over the relevant concepts. It is in theoretical computational complexity which utilizes concepts like, "languages of infinite many words", and the infinite is not existing in the computer practice (on the contrary some computer practitioners may consider it a computer worm!). So when I started studying the P vs. NP problem, the first that I asked myself was, **"From which axioms, should I start reasoning?"** Soon I realized that I should start reasoning from the axioms of the mathematical set theory.

We must notice here that the P versus NP problem is in fact a set of different problems when they are in the context of different axiomatic systems of set theory. In the context of what axiomatic system is the Complexity Theory of Turing machines? Since the complexity theory of Turing machines requires entities like infinite sets of words etc. and classes of them, then it is in the context of some standard axiomatic system of the mathematical set theory, which must include the axiom of infinite. So we notice that the next are different versions of the P vs. NP problem:

1) The P versus NP problem in a standard axiomatic system of set theory with the axiom of infinite and without the axiom of choice and this axiomatic system formulated e.g. in 1st-order or 2nd-order formal languages.

2) The P versus NP problem in an axiomatic system of set theory which includes the axiom of choice and the axiom of infinite and this axiomatic system is formulated in a 1st order or 2nd order formal languages.

3) Etc.

About this with references for different axiomatic systems of the mathematical set theory, we will talk again in the paragraph 2.

We notice also the P versus NP problem.

1) It is a difficult problem, that has troubled the scientific community for some decades.

2) It may have simple proofs of a few paragraphs, hopefully not longer than the proof of the Time Hierarchy theorem, which seems to be a deeper result.

3) But it can also have very lengthy and complex proofs, that may take dozens of pages.

What this proof is or is not:

1) It does not introduce new theoretical concepts in computational complexity theory so as to solve the P versus NP.

2) It does not use relativization and oracles.

3) It does not use diagonalization arguments, although the main proof, utilizes results from the time hierarchy theorem.

4) It is not based on improvements of previous bounds of complexity on circuits.

5) It is proved with the method of counter-example. Thus it is transparent short and "simple". It takes any EXPTIME-complete language decided by a Deterministic Turing machine (DTM) and reformulates it so that it is apparent that it belongs in the NP complexity class while it does not belong to the P complexity class of languages (For the definitions of the terms see the paragraph 2).

6) It seems like a "simple" proof because it chooses the right context to make the arguments with the key abstraction mentioned above. So it helps the scientific community to accept that this 3rd Clay Millennium problem has already been solved.

When we say "in the context of Deterministic Turing Machines" we mean that we do not involve non-Deterministic Turing machines as was originally the formulation of the complexity class NP. (For the definitions of the terms see paragraph 2).

In paragraph 4, we give an advanced, full proof of short length that $P \neq NP = EXPTIME$, in the standard context of deterministic Turing machines, solving thus the 3rd Clay Millennium problem.

2. Preliminary Concepts, and the Formulation of the 3rd Clay Millennium Problem, P versus NP

The theory of computational complexity, belongs to computer science, but it uses concepts like a set of words, infinite sets of words, etc., therefore it is a mathematical theory as well. It is not an independent mathematical axiomatic system, but it belongs to the mathematical axiomatic system of set theory which is used as the foundation of mathematics. The most popular axiomatic system for the standard mathematical set theory is that of Zermelo-Frankel (see Frankel A. A. [9] and Wikipedia [10]). This axiomatic system does not utilize classes only sets. If we want to include classes (e.g. the class of all sets) then we should use the Neumann-Bernays-Goedel axiomatic system for the set theory (see Jech T. 1978 [11] or Wikipedia [12]). Since in complexity theory, we refer freely to totalities of sets (languages as sets) without always strict predicates or functions over other sets, then the Neumann-Bernays-Goedel axiomatic system of set theory that allows classes might be more convenient. Therefore, any standard solution of the P vs. NP problem should be considered to exist in the context of the axiomatic system of Neumann-Bernays-Goedel for the mathematical set theory. But also alternatively in the context of the axiomatic system of Zermelo-Frankel, of set theory. And if we want also a version of mathematical formal logic where the arguments are written, this could also be the 2nd-order formal languages of Logic (see [13] Manin Y. I. 2010 or Wikipedia [14]).

In this paragraph, for the sake of the reader, we will mention only the basics to understand the formulation of the 3rd Clay Millennium problem. The official formulation is found in [15] (Cook, Stephen (*April* 2000), *The P versus NP*

Problem (*PDF*), *Clay Mathematics Institute site*). Together with an appendix where there is a concise definition of what are the **Deterministic Turing machines (**in short **DTM)**, that is considered that they formulate, in Computational Complexity theory, the notion and ontology of the software computer programs that a computer can run, in other words, **computer algorithms** (see also Wikipedia [16]).

There is also the concept **of non-Deterministic Turing machine (NDTM or NTM)** (See also [17] John C. Martin (1997). and [18] Papadimitriou Christos (1994) and [19] Wikipedia). Roughly speaking a non-Deterministic Turing Machine (NDTM) misses some automatic decisions (and requires possibly a user to decide for it) thus after a step it may have more than one possible next step without specifying in a deterministic way which one. This is essentially the case with **"wizards"** and **user-interactive** on-the-screen computer algorithms. When in the discussion and arguments we involve only Deterministic Turing machines we say that we are in the **context of Deterministic Turing machines**.

In the same paper are also defined the computational complexity ***classes P and NP.***

In computer science, the **computational complexity** or simply **complexity** of an algorithm is the amount of resources required to run it. We focus in particular on computation time or run-time (generally measured by the number of needed elementary operations) and memory storage requirements. The complexity of a language (problem) is the complexity of the best algorithms (least complexity) that allow solving the problem (see [17] John C. Martin (1997). and [18] Papadimitriou Christos (1994)). Because computational complexity as we shall see below is defined with the big O notation of mathematical functions, strictly speaking for general types of languages decided by a Turing machine, for the definition of its least complexity, to be correct, a proof should be provided for the existence of such a "least" complexity.

The **run-time complexity** of a computer algorithm (Deterministic Turing machine) is defined and symbolized as **DTIME(*f*(*n*))** where f is a function of the natural number n iff given initial data of size n, the algorithm (Deterministic Turing machine) terminates (decides) within $f(n)$ steps. (See also [17] John C. Martin (1997). and [18] Papadimitriou Christos (1994) and [20] [21] Wikipedia) This is also expressed by saying that the duration of the computation in steps, is of the order $O(f(n))$. Where by $O(f(n))$ is meant (see Wikipedia [22] [16]) that if $T(n)$ is the time that the Turning machines terminate as a function of the size n of the initial data, then $T(n) \leq M^* f(n)$ for some positive constant M not depending on n, for all n, thus as $n \to +\infty$. By **DTIME(*f*(*n*))** is also denoted the class of all languages that are decidable by a Deterministic Turing machine in $O(f(n))$ run time.

The computational complexity ***class P*** is defined as the class of languages that are decided by deterministic Turing machines with run-time polynomial complexity. In symbols **P = PTIME** is the class of languages decided by a deterministic

Turing machine that runs for some polynomial p in complexity $\mathbf{DTIME}(p(|n|))$, the polynomial depending on the language. (See also [23] Wikipedia). In other symbols.

$$\mathrm{P} = \mathbf{U}_{k\in N}\,\mathbf{DTIME}(n^k)$$

The elements of the classes P, NP, etc., strictly speaking are not only sets of words denoted by L, that is not only languages, but also for each such set of words or language L at least one Deterministic Turing machine (DTM), M that decides it, in a specified complexity so they are pairs (L, M). Two such pairs (L_1, M_1) (L_2, M_2) are also ***equidecidable*** in other words $L_1 = L_2$ although it may happen that $M_1 \neq M_2$. The complexity of the language is considered the least complexity, if it exists, that decides it. E.g. if the complexity of M_1 is polynomial-time while that of M_2 is exponential-time choosing the first pair instead of the second means that we have turned a high-complexity problem into a feasible low-complexity problem.

The definition of other computational complexity classes like **EXPTIME** etc. can be found in standard books like [17] [18] [24].

In computational complexity theory, the complexity class **EXPTIME** (sometimes called **EXP** or **DEXPTIME**) is the class of all languages that are decidable by a deterministic Turing machine in exponential time, *i.e.*, in $O(2^{p(|n|)})$ time, where $p(|n|)$ is a polynomial function of $|n|$. In symbols.

$$\mathrm{EXPTIME} = \mathbf{U}_{k\in N}\,\mathbf{DTIME}(2\hat{}(n^k))$$

(See also Wikipedia [25]).

In the official formulation [3] there is also the definition of the concept of ***a decision problem language in polynomial time reducible to another decision problem language***.

Based on this definition it is defined that an EXPTIME-complete decision language of EXPTIME is EXPTIME-complete, when all other decision problem languages of EXPTIME have a polynomial time reduction to it. Here is the exact definition. We denote by Σ^* *all the words of an alphabet* Σ.

Definition 2.1. *Suppose that L_i is a language from words of Σ^*_i, $i = 1, 2$. Then $L_1 \leq pL_2$ or $L_1 \leq polyL_2$ (L_1 is polynomially p-reducible to L_2) if and only if there is a polynomial-time computable function-map* (*or total function*) $f : \Sigma_1^* \to \Sigma_2^*$ (*in other words* $(x, f(x)) \in f$ *is in polynomial time decidable*) *such that* $x \in L_1$ *if and only if* $f(x) \in L_2$, *for all* $x \in \Sigma_1^*$.

Lemma 2.1. If $L_1 \leq polyL_2$ *by the polynomial time decidable function f, then* $|f(x)| \leq p(|x|)$ *for some polynomial p.*

Remark 2.0. If The function f is also on-to (surjective) it holds also the reverse inequality, that is there is a polynomial q with $|x| \leq q(|f(x)|)$. Here is why. Let f:A -> B be a computable function with domain (f) = A and range (f) a subset of B, We say that the f is computably invertible if there is a computable function g:B -> A, such that with input y, it gives output x, such that $f(x) = y$ or it give output # if it does not exist x with $f(x) = y$ (y is outside the range of f). It holds that a computable function f as above is computably invertible if and only if the

range (f) is decidable. Thus if range (f) = B, the computable f is always computably invertible, and if the f is of polynomial complexity, the same proof gives that its inverse g is also of polynomial complexity. Thus there is a polynomial q with $|x| \leq q(|f(x)|)$.

In the same papers or books [15] [17] [24] [26] can be found the concepts and definitions of ***NP-complete and EXPTIME-compete decision problems***. See also [27], [18] where it's proved that specific decision problems are EXPTIME-complete. E.g. in [15] in Definition 4, it is defined that a language of the complexity class NP is NP-complete if and only if any other language of the class NP has a polynomial reduction to it.

In particular, also it holds that,

Lemma 2.2. *If the language L_c of the complexity class C (=P, NP, EXPTIME, etc.) is a C-complete language, (C = P, NP, EXPTIME, etc.) then any other language L of C has a polynomial reduction onto the language L_c.*

For simplicity, we will consider here only binary alphabets {0, 1} and sets of binary words Σ.

Since we are obliged to take strictly the official formulation of the problem, rather than textbooks about it, we make the next clarifications.

We will use the next conditions for a Language to be in the class NP, as stated in the official formulation of the P versus NP problem (see [15] Cook, Stephen (*April* 2000), *The P versus NP Problem* (*PDF*), *Clay Mathematics Institute*.).

We denote by Σ^* *all the words of an alphabet* Σ.

Definition 2.2. *A language L of binary words is in the class NP if and only if the next conditions hold.*

1) *There is a deterministic Turing machine M that decides L. In other words for any word x in L, when x is given as input to M, then M accepts it and if x does not belong to L then M rejects it.*

In symbols: $\exists$ *a deterministic Turing machine M, such that* $\forall x \in \Sigma^*$, *x is either accepted or rejected by M and if M accepts* $x \rightarrow x \in L$, *and if M reject* $x \rightarrow x$ *does not belong to L.*

2) There is a polynomial-time checkable relation $R(x, y)$ which as set of pairs of words can also be considered a polynomial decidable language R, and a natural number k of N, so that for every word x, x belongs to L if and only if there is a word y, with $|y| \leq |x|^k$, and $R(x, y)$ holds or equivalently (x, y) belongs to R.

In symbols: $\exists$ *relation* (*language*) *R which is polynomial-time checkable* (*polynomial complexity decidable language R*), *and* $\exists k \in N$, *such that* $\forall x \in \Sigma^*$, $x \in L \leftrightarrow (\exists y \in \Sigma^*, |y| \leq |x|^k$ *and* $(x, y) \in R)$. *Or equivalently.*

$$L = \{x / x \in \Sigma^* \text{ and } \exists y \in \Sigma^*, |y| \leq |x|^k \text{ and } (x, y) \in R\}.$$

3) *The complexity class NP is contained in the complexity class EXPTIME.*

Remark 2.1. In the official statement of the P versus NP problem (see [15] Cook, Stephen (*April* 2000), *The P versus NP Problem* (*PDF*), *Clay Mathematics Institute*) condition 1) is not mentioned. But anyone that has studied complexity theory, knows that it holds. The languages of NP cannot be semidecidable (or

undecidable). The NP class is also defined as NP = $\bigcup_{k\in N}$ NTIME(n^k), but this definition is in the **context of non-deterministic Turing Machines.** This means that we involve non-deterministic Turing machines.

Remark 2.2. Notice that in condition 2) the k depends on the relation R and is not changing as the certificate y changes. In other words, k does not depend on y and we ***did not*** state the next:

There is a polynomial-time checkable relation $R(x, y)$, so that for every word x, x belongs to L if and only if there is a word y, and k in N, with $|y| \le |x|^k$, and $R(x, y)$ holds. In symbols: $\exists$ *relation R which is polynomial-time checkable, such that* $\forall x \in \Sigma^*$, $x \in L \leftrightarrow (\exists y \in \Sigma^*$ *and* $\exists k \in N$ *such that* $|y| \le |x|^k$ *and* $R(x, y)$ *holds*).

In the official statement of the P versus NP problem (see [15] Cook, Stephen (*April* 2000), *The P versus NP Problem* (*PDF*), *Clay Mathematics Institute*) this is not made clear, in the natural language that the definition is stated. But that k does not depend on the certificate, but on the polynomial checkable relation becomes clear, when we look at the proof in any good textbook about complexity theory, of how a non-deterministic Turing machine which runs in polynomial time, can define a deterministic Turing machine with a polynomial time checkable relation, which is considered that replaces it.

More generally modern formulations of the definition 2.2 instead of integer k such that $|y| \le |x|^k$ require a polynomial $p(|x|)$ so that $|y| \le p(|x|)$.

Remark 2.3. Usually, condition 3) in definition 2.2 is not stated. But if for each word x of L, we take all possible worlds y of Σ^*, of length $\le |x|^k$ and check in polynomial time complexity if $R(x, y)$ holds or not, we result in the worst case scenario to an exponential time complexity algorithm that decides the language L.

3. Well-Known Results That Will Be Used

We will not use too many results from the computational complexity theory for our proof that P ≠ NP.

A very deep theorem in Computational Complexity is the ***Time Hierarchy Theorem*** (see e.g. [17] [18] [24] [26] [28]. This theorem gives the existence of decision problems that cannot be decided by any other deterministic Turing machine in less complexity than a specified.

Based on this theorem, it is proved that:

Proposition 3.1. *There is at least one EXPTIME-complete decision language* (*problem*), *that cannot be decided in polynomial time, thus P ≠ EXPTIME.*

The next two propositions indicate what is necessary to prove in order to give the solution to the P versus NP problem.

Proposition 3.2. *If the class NP contains a language L which cannot be decided with a polynomial time algorithm, then P ≠ NP.*

Proposition 3.3. *If the class NP contains a language L which is EXPTIME complete, then NP = EXPTIME.*

4. The Solution: P ≠ NP = EXPTIME in the Context of Deterministic Turing Machines (DTM)

We will prove in this paragraph that P ≠ NP is the in the context of second-order formal language of mathematical set theory.

The strategy to solve the P vs. NP problem in the current paper (and in my previous publications) is by starting with an EXPTIME-complete language (problem), that proposition 3.1 guarantees that it exists, and proving that it has a re-formulation as an NP-class language with verifier relation and certificate, thus NP = EXPTIME. Of course in order to proceed like this we must not share the belief that the complexity class NP is a strict subclass of the complexity class EXPTIME otherwise we would not think to start like this.

Definition 4.1. *A language L, which is Turing-machine decidable and of complexity at most exponential is called* ***a passwords language*** *if for each length l of words, there is at most one word x in the language of length l,* $|x| = l$. *If the language L is also an EXPTIME-complete language, then it is called* ***an enigma passwords language.***

The strategy to find one is quite simple: We will start with an exptime-complete decision problem and its language L_{exp} and we will derive from it **an enigma passwords language** and then **prove that the passwords language belongs to the complexity class NP.** Thus an NP class decision problem cannot be solved in the polynomial time (it does not belong to the class P).

The next proposition sets the existence of an EXPTIME-complete complexity language of the EXPTIME complexity class (Proposition 3.1) in a convenient form, that can be used for further reasoning.

Proposition 4.1. *There is at least one infinite binary sequence, that can be computed - decided in exptime-complete complexity. And there is an enigma passwords language. In other words, an exptime-complete complexity language, such that for each length of word n it has exactly one word in it of length n.*

Proof.

Let an exptime-complete language L_{exp}, that its existence is guaranteed by **Proposition 3.1**. If Σ^* is the set of all words of the binary alphabet Σ of the language L_{exp}, then we give a linear order to the binary alphabet $\Sigma = \{0, 1\}$ $0 < 1$, and then the linear lexicographic order to the set of all words of $\Sigma^* = \{w_0, w_1, \ldots, w_n \ldots\}$. Since L_{exp} is a subset of Σ^* it inherits the linear order of its words. So let $\text{Char}(L_{exp}, \Sigma^*)$: Σ^*. -> {0, 1} be the characteristic function of the set L_{exp} in Σ^*. Then the values of the $\text{Char}(L_{exp}, \Sigma^*)$ consist of binary digit, that are equal to 0 or 1, denoted by d_i, for $i \in \boldsymbol{N}$, and i being the above-mentioned linear sequential ordering of Σ^*. This binary sequence d_i, for $i \in \boldsymbol{N}$ is obviously of exptime-complete complexity because for each i, to define the d_i, requires a decision if the ith word of Σ^*. belongs to L_{exp} or not. A first finite 7-digits segment of it, would seem for example like (0010110...). We denote this exptime-complete binary sequence by DNA_{exp}. In the next, we take the language $L_{pe} = \{w_1 \ldots, w_n \ldots\}$ which consists, for each n from the first n binary digits w_n of the DNA_{exp}. Again, obviously, this

language is exptime-complete too, and furthermore, for each word length n, it has exactly one only word $|w_n| = n$ of length n in it. Thus L_{pe} is an enigma passwords language. In the previous example of the 7 first digits of DNA_{exp} (0010110...) the enigma passwords language would include the 7 first passwords L_{pe} = {0, 00, 001, 0010, 00101, 001011, 0010110…} QED.

Remark 4.1: The gradual explicit certified publication of an enigma language of passwords.

In the next, we introduce the action of two different Turing machines. Let us take again the Enigma passwords language L_{pe} and the exptime-complete complexity Turing machine which decides it, denote by TM_e and called **Engima Turing machine**. The name Enigma is obviously from the heavy encryption machine that the Germans utilized during the 2nd World War. We use this Turing machine to enumerate the passwords of the L_{pe} and create certificates for them. In other words, the TM_e scans the linearly ordered sequence of all the words of Σ^* as in the proof of the previous proposition 4.1 and decides if a word is a password of the enigma passwords language L_{pe}. If at time t the nth word w_n is a password of L_{pe}. it creates a certificate denoted by c_n which is a word equal to w_n, $c_n = w_n$ and writes it in a list of certificates denoted by $L_{c,n} = \{c_0, c_1, \ldots, c_n\}$ which are the certified passwords till the word length n. Given sufficient time t, any password of L_{pe}. of word-length at most m will have a certificate in $L_{c,m} = \{c_0, c_1, \ldots, c_m\}$. We call this process of enumeration by the enigma Turing machine as **the gradual explicit certified publication of an enigma language of passwords.** In the next comes the action of a second different and lighter Turing machine denoted by TM_c called the **certifier Turing machine.** This Turing machine takes as input any word w_n of Σ^* of length n, and scans the published list of certificates $L_{c,m} = \{c_0, c_1, \ldots, c_m\}$ $m \geq n$ created by the Enigma Turing machine, to discover if there is a certificate $c_n = w_n$. So from the length n of w_n it goes directly to the nth place of $L_{c,m}$ (which takes only polynomial time based on n) and then checks if $c_n = w_n$ or not (which takes again only polynomial time based on n). If yes, the certifier Turing machine TM_c claims that w_n belongs to L_{pe}. We denote this action of TM_c with input a word w_n to find a certificate c_n in $L_{c,m}$ of length n and check if $c_n = w_n$ as Relation $R(c_n, w_n)$. Therefore the action of the **certifier Turing machine** TM_c which is of **polynomial complexity** can be described in logical symbols

$\exists$ *relation R which is polynomial-time checkable* (*polynomial complexity decidable R by* TM_c), *and* $\exists k \in N$, *such that* $\forall x \in \Sigma^*$, $x \in L_{pe} \leftrightarrow (\exists y \in \Sigma^*, |y| \leq |x|^k$ *and* $(x, y) \in R)$. *Or equivalently.*

$L_{pe} = \{x / x \in \Sigma^*$ *and* $\exists y \in \Sigma^*, |y| \leq |x|^k$ *and* $(x, y) \in R\}$. Here of course $k = 1$.

Nevertheless, this is exactly the condition 2) in the definition 2.2 of an NP-class language, thus L_{pe} is an NP-class language.

Proposition 4.2 (3rd Clay Millennium problem 1st solution) *There is at least one language of the class NP which is also an exptime-complete language of EXPTIME, thus NP = EXPTIME and therefore P ≠ NP*

Proof.

From the arguments in the previous Remark 4.1, the Enigma passwords language L_{pe} is exptime-complete after the combined action of the Enigma Turing machine $\boldsymbol{TM_e}$ and the certifier Turing machine $\boldsymbol{TM_c}$ in the process of **gradual explicit certified publication of an enigma language of passwords** L_{pe} proves this language L_{pe}, after the action of the certifier Turing machine $\boldsymbol{TM_c}$ to be of the type of NP-class with certifiers. Therefore NP = EXPTIME and from the hierarchy theorem consequently P ≠ NP. *QED.*

In the next, we provide a 2nd solution to the millennium problem.

There is a hidden similarity or affiliation between Definition 2.2 of the NP complexity class with polynomial complexity verifier relations and the definition of polynomial complexity reduction of a language to another language Definition 2.1. Part of this similarity or affiliation is revealed in the next lemma.

Lemma 4.1. ***Polynomial reduction of languages and the complexity class NP.***

Let a Turing-machine decidable Language L over the alphabet Σ. *in the complexity class EXPTIME.*

If there is polynomial reduction of the full language Σ^* *on L* (*surjective*), $\Sigma^* \leq$ *poly L then the language L is in the complexity class NP.*

Proof: From the definition 2.1 since $\Sigma^* \leq$ *poly L* there is a polynomial time computable function-map R: $\Sigma^* \to \Sigma^*$, such that y belongs to Σ^* if and only if $R(y)$ belongs to L. Furthermore from the Lemma 2.1 and Remark 2.0, $|R(x)| \leq p(|x|)$ for some polynomial p. Or to rephrase it, $(y, R(y)) \in R$ is polynomial time decidable $|y| \leq p(|x|)$ and y belongs to Σ^* if and only if and $x = R(y)$ belongs to L. Reversing the logical equivalence relation: $x = R(y)$ belongs to L if and only if y belongs to Σ^* (that is it is a word over the alphabet Σ). (and R and $|y| \leq p(|x|)$ as above).

Again rephrasing it: $(y, x) \in R$ is polynomial time decidable, $|y| \leq p(|x|)$ for some polynomial p (depending on R and not on x), and x belongs to L if and only if there is a word y of Σ^* such that $(y, x) \in R$. Or in symbols, ∃ relation (language) R which is polynomial complexity decidable language, and ∃ polynomial p such that, $x \in L \leftrightarrow (\exists y \in \Sigma^*$, with $|y| \leq p(|x|)$ and $(x, y) \in R)$. But this is just the condition 2) of the definition 2.2 of the complexity class NP, where the relation, $|y| \leq |x|^k$ has been substituted by the $|y| \leq p(|x|)$ as at the end of Remark 2.2. The language L is furthermore decidable by a Turing machine and at most of EXPTIME complexity, thus conditions 1) and 3) of Definition 2.2 are also satisfied, consequently the language L is of the type of NP-complexity class. QED.

We could explore further the similarity and affiliation of the concepts of polynomial complexity verifier relation with certificates of a language L in the NP-class, by proving that if the verifier relation R is also a function map from the language of certificates $C(L)$ on to L, then there is also polynomial complexity reduction of the language of certificates $C(L)$ to the original language L: $C(L) \leq$ poly L. But we do not need to do so for the purpose of solving the P vs. NP problem.

The Remark 4.1 may seem simple to many and certainly, for me it was obvious, that is why I did not include in my previous publications of the solution of the P vs. problem [4] [5] [6] [8] [29] Kyritsis K. But strictly speaking without Remark 4.1, the previously published solutions by me of the P vs. NP are not fully and in a very detailed way understandable. Lemma 4.1 requires lucky intuitive thinking and it's a key perception for the solution of the P vs. NP problem.

We proceed to solve the P vs. NP problem in a slightly different and 2nd way.

Proposition 4.3. (2nd solution of the 3rd Clay Millennium problem P vs. NP) *There is at least one* (*decision problem*) *language of the class NP which is not also in the class P. Therefore*, $P \neq NP$. *It holds furthermore that* $NP = EXTIME$.

Proof: We start with the Enigma language of passwords L_{pe} of proposition 4.1 We know that there it is exptime-complete, thus all other languages of EXPTIME will have a polynomial reduction on it. Therefore the language Σ^* of all words over the alphabet Σ, which is polynomial time complexity decidable also belongs to EXPTIME, $\Sigma^* \in$ EXPTIME. Thus $\Sigma^* \leq poly\ L_{pe}$. We are interested in particular for a surjective (on-to) reduction of $\Sigma^* \leq poly\ L_{pe}$. Here is how we can be sure that there exists one. We define the reduction *f*. We will utilize the process of the **gradual explicit certified publication of the enigma language of passwords** L_{pe} described in Remark 4.1. For any word w_n of Σ^* of length n, the certifier Turing machine TM_c scans the published list of certificates $L_{c,m} = \{c_0, c_1, \ldots, c_m\}$ $m \geq n$ created by the Enigma Turing machine TM_e, to find the certificate c_n and then maps the w_n on. c_n by the function *f*. This action is of polynomial complexity based on n. It is obviously a surjective polynomial reduction of $\Sigma^* \leq poly\ L_{pe}$ But then from the previous Lemma 4.1, we conclude that $L_{pe} \in$ NP-class! Therefore NP = EXPTIME. The hierarchy theorem concludes that P ≠ NP. Q.E.D.

5. Conclusions

The literature on the complexity theory has an abundance of problems or languages that are proven to be NP-complete, but a scarcity of languages or problems that are proven to be EXPTIME-complete.

The fact that none of the languages or problems that are known to be NP-complete have so far known algorithms to solve-decide them that in the general case are never less than exponential time, should ring a bell to the researchers!

Could it hold that all these NP-complete problems are also EXPTIME-complete problems? The current solution of the P vs. NP problem proves exactly this! The next proposition answers it in the affirmative and thus we have now an abundance of well-studied problems that turn out to be also EXPTIME-complete. This is a significant contribution to the theory of EXPTIME-complete languages or problems

Proposition 5.1. *All the NP-complete languages are also EXPTIME-complete languages*.

Proof: Direct from the equality NP = EXPTIME in propositions 4.2, or 4.3 QED.

There are many more consequences of the equality **NP = EXPTIME** that the specialist of complexity theory can derive without too much effort. For example, in [30] Mathoverflow it is discussed that if NP = EXPTIME then every Deterministic Turing Machine has a succinct "execution proof".

Sometimes great problems have relatively short and elegant solutions provided we find the **key abstractions** and convenient context, symbols and semantics to solve them. It requires also a certain power of thinking (especially when there are biased beliefs prohibiting thinking) rather than a complexity of thinking, in areas where traditionally and collectively it may not have existed before because of false dominating beliefs. Here the key-abstraction was to start from the class EXPTIME and an EXPTIME-complete language of it, without specifying which one instead of starting from the class NP. Then prove that it can be reformulated as an NP-class language. Since in my opinion, the Hierarchy Theorem is a deeper result than the P versus NP problem, in principle, there should exist a not much more complicated proof of the P versus NP problem, compared to the proof of the Hierarchy Theorem. Intuitively since ***the Non-deterministic Turing machines are like user-interactive algorithms*** that involve the ***free human will***, it is expected that the non-deterministic polynomial-time algorithms cannot be computed with less than exponential time complexity. This includes encryption and passwords setting problems.

The proof of the P versus NP problem in the direction P ≠ NP, also means that the standard practice of passwords setting in the internet is safe when the encryptions is not corrupted and the publicly available hardware computational power is the same for all.

There are many students who are surprised by the "difficulty" of the P vs. NP problem and ask why the P vs. NP problem was not solved long ago, by proving that the **password-breaking** cannot be done in polynomial time but only in exponential time. Actually, this is exactly what we proved! Only the theoretical formulation of encryption of finite many of passwords is essentially similar to the presentation of an infinite many words, and languages in an abstract way without specifying it, except of general requirements like being EXPTIME-complete as we did in the arguments of the current paper.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Woeginger, G.J. (2016) The P versus NP Page. https://www.win.tue.nl/~gwoegi/P-versus-NP.htm

[2] Wikipedia, "P versus NP Problem". https://en.wikipedia.org/wiki/P_versus_NP_problem

[3] Yannakakis, M. (1998) Expressing Combinatorial Optimization Problems by Linear Programs. *Proceedings of STOC*, Chicago, 2-4 May 1988, 223-228. https://doi.org/10.1145/62212.62232

[4] Kyritsis, C. (2017) On the Solution of the 3rd Clay Millennium Problem. A Short and Elegant Proof That P ≠ NP in the Context of Deterministic Turing Machines and Zermelo-Frankel Set Theory. *Proceedings of the* 1*st ICQSBEI* 2017 *Conference*, Athens, Greece, 170-181.

[5] Kyritsis, K. (2018) The Solution of the 3rd Clay Millennium Problem. A Short Proof That P ≠ NP = Exptime in the Context of Zermelo Frankel Set Theory. *International Journal of Pure and Applied Mathematics*, **120**, 497-510. http://www.ijpam.eu

[6] Kyritsis, K. (2021) Study on the Solution of the Clay Millennium Problem about the P vs. NP: A Short Proof that P not equal NP = EXPTIME in the Context of Deterministic Turing Machines. In: *New Visions in Science and Technology*, Vol. 6, BP International, India, 60-69. https://doi.org/10.9734/bpi/nvst/v6/5176F https://stm.bookpi.org/NVST-V6/article/view/4135

[7] Kyritsis, K. (2022) A Short and Simple Solution of the Millennium Problem about the Navier-Stokes Equations and Similarly for the Euler Equations. *Journal of Applied Mathematics and Physics*, **10**, 2538-2560. https://www.lap-publishing.com/catalog/details//store/gb/book/978-620-4-72562-8/the-solutions-of-the-3rd-and-4th-millennium-mathematical-problems

[8] Kyritsis, K. (2023) A 3rd Shorter Solution of the Clay Millennium Problem about P ≠ NP = EXPTIME. *Conference*: 6*th International Conference on Quantitative*, *Social*, *Biomedical and Economic Issues*, Athens, 1 July 2022, 81-89. https://icqsbei2022.blogspot.com/2022/06/blog-post.html http://books.google.com/books/about?id=xZnCEAAAQBAJ

[9] Frankel, A.A. (1976) Abstract Set Theory. North Holland Publishing Company, Amsterdam.

[10] Wikipedia, "Zermelo Frankel Set Theory". https://en.wikipedia.org/wiki/Zermelo%E2%80%93Fraenkel_set_theory

[11] Thomas, J. (1978) Set Theory. Academic Press, Cambridge.

[12] Wikipedia "Von Neumann-Bernays-Gödel set theory" https://en.wikipedia.org/wiki/Von_Neumann%E2%80%93Bernays%E2%80%93G%C3%B6del_set_theory

[13] Manin, Y.I. (2010) A Course in Mathematical Logic for Mathematicians. Springer, Berlin. https://doi.org/10.1007/978-1-4419-0615-1

[14] Wikipedia, "Second-order Logic". https://en.wikipedia.org/wiki/Second-order_logic

[15] Cook, S. (2000) The P versus NP Problem (PDF), Clay Mathematics Institute Site.

[16] Wikipedia, "Turing Machine". https://en.wikipedia.org/wiki/Turing_machine

[17] Martin, J.C. (1997) Introduction to Languages and the Theory of Computation. 2nd Edition, McGraw-Hill, London.

[18] Papadimitriou, C. (1994) Computational Complexity. Addison-Wesley, Boston.

[19] Wikipedia, "Nondeterministic Turing Machine". https://en.wikipedia.org/wiki/Nondeterministic_Turing_machine

[20] Wikipedia, "DTIME". https://en.wikipedia.org/wiki/DTIME

[21] Wikipedia, "Time Complexity". https://en.wikipedia.org/wiki/Time_complexity#Polynomial_time

[22] Wikipedia "Big O Notation". https://en.wikipedia.org/wiki/Big_O_notation

[23] Wikipedia, "P (Complexity)". https://en.wikipedia.org/wiki/P_(complexity)

[24] Lewis, H.R. and Papadimitriou, C.H. (1981) Elements of the Theory of Computation. Prentice-Hall, Englewood Cliffs.

[25] Wikipedia, "EXPTIME". https://en.wikipedia.org/wiki/EXPTIME

[26] Trevisan, L. (2009) Notes on Hierarchy Theorems. University of California, Berkeley.

[27] Hartmanis, J. and Stearns, R.E. (1965) On the Computational Complexity of Algorithms. *Transactions of the American Mathematical Society*, **117**, 285-306. https://doi.org/10.1090/S0002-9947-1965-0170805-7

[28] Stanislav, Ž. (1983) A Turing Machine Time Hierarchy. *Theoretical Computer Science*, **26**, 327-333. https://doi.org/10.1016/0304-3975(83)90015-4

[29] Kyritsis, K. (2021) Review of the Solutions of the Clay Millennium Problem about P ≠ NP = EXPTIME. *World Journal of Research and Review*, **13**, 21-26. https://www.wjrr.org/vol-13issue-3 https://doi.org/10.31871/WJRR.13.3.8

[30] Mathoverflow. If NP = EXPTIME, Does Every DTM Have a Succinct "Execution Proof"? https://mathoverflow.net/questions/114887/if-np-exptime-does-every-dtm-have-a-succinct-execution-proof