



## NP on Logarithmic Space

---

Frank Vega

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

August 6, 2023

# NP on Logarithmic Space

Frank Vega <sup>\*1</sup>

## Abstract

$P$  versus  $NP$  is considered as one of the most important open problems in computer science. This consists in knowing the answer of the following question: Is  $P$  equal to  $NP$ ? It was essentially mentioned in 1955 from a letter written by John Nash to the United States National Security Agency. However, a precise statement of the  $P$  versus  $NP$  problem was introduced independently by Stephen Cook and Leonid Levin. Since that date, all efforts to find a proof for this problem have failed. Another major complexity classes are  $L$  and  $NL$ . Whether  $L = NL$  is another fundamental question that it is as important as it is unresolved. We prove that  $NP \subseteq NSPACE(\log^2 n)$  just using logarithmic space reductions.

2020 MSC: MSC 68Q15, MSC 68Q17

KEYWORDS: Computational Algorithm, Complexity Classes, Completeness, Polynomial Time, Reduction, Logarithmic Space

## 1 Introduction

In 1936, Turing developed his theoretical computational model [10]. The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to this theoretical model for computation [10]. A deterministic Turing machine has only one next action for each step defined in its program or transition function [10]. A nondeterministic Turing machine could contain more than one action defined for each step of its program, where this one is no longer a function, but a relation [10].

Let  $\Sigma$  be a finite alphabet with at least two elements, and let  $\Sigma^*$  be the set of finite strings over  $\Sigma$  [2]. A Turing machine  $M$  has an associated input alphabet  $\Sigma$  [2]. For each string  $w$  in  $\Sigma^*$  there is a computation associated with  $M$  on input  $w$  [2]. We say that  $M$  accepts  $w$  if this computation terminates in the accepting state, that is  $M(w) = \text{"yes"}$  [2]. Note that,  $M$  fails to accept  $w$  either if this computation ends in the rejecting state, that is  $M(w) = \text{"no"}$ , or if the computation fails to terminate, or the computation

ends in the halting state with some output, that is  $M(w) = y$  (when  $M$  outputs the string  $y$  on the input  $w$ ) [2].

Another relevant advance in the last century has been the definition of a complexity class. A language over an alphabet is any set of strings made up of symbols from that alphabet [4]. A complexity class is a set of problems, which are represented as a language, grouped by measures such as the running time, memory, etc [4]. The language accepted by a Turing machine  $M$ , denoted  $L(M)$ , has an associated alphabet  $\Sigma$  and is defined by:

$$L(M) = \{w \in \Sigma^* : M(w) = \text{“yes”}\}.$$

Moreover,  $L(M)$  is decided by  $M$ , when  $w \notin L(M)$  if and only if  $M(w) = \text{“no”}$  [4]. We denote by  $t_M(w)$  the number of steps in the computation of  $M$  on input  $w$  [2]. For  $n \in \mathbb{N}$  we denote by  $T_M(n)$  the worst case run time of  $M$ ; that is:

$$T_M(n) = \max\{t_M(w) : w \in \Sigma^n\}$$

where  $\Sigma^n$  is the set of all strings over  $\Sigma$  of length  $n$  [2]. We say that  $M$  runs in polynomial time if there is a constant  $k$  such that for all  $n$ ,  $T_M(n) \leq n^k + k$  [2]. In other words, this means the language  $L(M)$  can be decided by the Turing machine  $M$  in polynomial time. Therefore,  $P$  is the complexity class of languages that can be decided by deterministic Turing machines in polynomial time [4]. A verifier for a language  $L_1$  is a deterministic Turing machine  $M$ , where:

$$L_1 = \{w : M(w, u) = \text{“yes” for some string } u\}.$$

We measure the time of a verifier only in terms of the length of  $w$ , so a polynomial time verifier runs in polynomial time in the length of  $w$  [2]. A verifier uses additional information, represented by the string  $u$ , to verify that a string  $w$  is a member of  $L_1$ . This information is called certificate.  $NP$  is the complexity class of languages defined by polynomial time verifiers [8].

It is fully expected that  $P \neq NP$  [8]. Indeed, if  $P = NP$  then there are stunning practical consequences [8]. For that reason,  $P = NP$  is considered as a very unlikely event [8]. Certainly,  $P$  versus  $NP$  is one of the greatest open problems in science and a correct solution for this incognita will have a great impact not only in computer science, but for many other fields as well [3]. Whether  $P = NP$  or not is still a controversial and unsolved problem [1]. We provide an important step forward for this outstanding problem using the logarithmic space complexity.

## 1.1 The Hypothesis

A function  $f : \Sigma^* \rightarrow \Sigma^*$  is a polynomial time computable function if some deterministic Turing machine  $M$ , on every input  $w$ , halts in polynomial

time with just  $f(w)$  on its tape [10]. Let  $\{0, 1\}^*$  be the infinite set of binary strings, we say that a language  $L_1 \subseteq \{0, 1\}^*$  is polynomial time reducible to a language  $L_2 \subseteq \{0, 1\}^*$ , written  $L_1 \leq_p L_2$ , if there is a polynomial time computable function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  such that for all  $x \in \{0, 1\}^*$ :

$$x \in L_1 \text{ if and only if } f(x) \in L_2.$$

An important complexity class is *NP-complete* [5]. If  $L_1$  is a language such that  $L' \leq_p L_1$  for some  $L' \in \text{NP-complete}$ , then  $L_1$  is *NP-hard* [4]. Moreover, if  $L_1 \in \text{NP}$ , then  $L_1 \in \text{NP-complete}$  [4]. A principal *NP-complete* problem is *SAT* [5].

A logarithmic space Turing machine has a read-only input tape, a write-only output tape, and read/write work tapes [10]. The work tapes may contain at most  $O(\log n)$  symbols [10]. In computational complexity theory,  $L$  is the complexity class containing those decision problems that can be decided by a deterministic logarithmic space Turing machine [8].  $NL$  is the complexity class containing the decision problems that can be decided by a nondeterministic logarithmic space Turing machine [8].

In general,  $DSPACE(S(n))$  and  $NSPACE(S(n))$  are complexity classes that are used to measure the amount of space used by a Turing machine to decide a language, where  $S(n)$  is a space-constructible function that maps the input size  $n$  to a non-negative integer [7]. The complexity class  $DSPACE(S(n))$  is the set of languages that can be decided by a deterministic Turing machine that uses  $O(S(n))$  space [7]. The complexity class  $NSPACE(S(n))$  is the set of languages that can be decided by a nondeterministic Turing machine that uses  $O(S(n))$  space [7].

A function  $f : \Sigma^* \rightarrow \Sigma^*$  is a logarithmic space computable function if some deterministic Turing machine  $M$ , on every input  $w$ , halts using logarithmic space in its work tapes with just  $f(w)$  on its output tape [10]. Let  $\{0, 1\}^*$  be the infinite set of binary strings, we say that a language  $L_1 \subseteq \{0, 1\}^*$  is logarithmic space reducible to a language  $L_2 \subseteq \{0, 1\}^*$ , written  $L_1 \leq_l L_2$ , if there is a logarithmic space computable function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  such that for all  $x \in \{0, 1\}^*$ :

$$x \in L_1 \text{ if and only if } f(x) \in L_2.$$

The logarithmic space reduction is used for the completeness of the complexity classes  $L$ ,  $NL$  and  $P$  among others.

We can give a certificate-based definition for  $NL$  [2]. The certificate-based definition of  $NL$  assumes that a logarithmic space Turing machine has another separated read-only tape, that is called “read-once”, where the head never moves to the left on that special tape [2].

**Definition 1.1.** *A language  $L_1$  is in  $NL$  if there exists a deterministic logarithmic space Turing machine  $M$  with an additional special read-once*

input tape polynomial  $p : \mathbb{N} \rightarrow \mathbb{N}$  such that for every  $x \in \{0, 1\}^*$ :

$$x \in L_1 \Leftrightarrow \exists u \in \{0, 1\}^{p(|x|)} \text{ then } M(x, u) = \text{“yes”}$$

where by  $M(x, u)$  we denote the computation of  $M$ ,  $x$  is placed on its input tape, the certificate string  $u$  is placed on its special read-once tape, and  $M$  uses at most  $O(\log |x|)$  space on its read/write tapes for every input  $x$  where  $|\dots|$  is the bit-length function. The Turing machine  $M$  is called a logarithmic space verifier.

We state the following Hypothesis:

**Hypothesis 1.2.** *There is a nonempty language  $L_2$  which is closed under logarithmic space reductions in  $NP$ -complete with a deterministic **square logarithmic** space Turing machine  $M$  using an additional special read-once input tape polynomial  $p : \mathbb{N} \rightarrow \mathbb{N}$ , where:*

$$L_2 = \{w : \exists u \in \{0, 1\}^{p(|w|)} \text{ such that } M(w, u) = \text{“yes”}\}$$

when by  $M(w, u)$  we denote the computation of  $M$ ,  $w$  is placed on its input tape, and the certificate string  $u$  is placed on the special read-once tape of  $M$ . In this way, there is a  $NP$ -complete language defined by a **square logarithmic** space verifier  $M$ .

We show the principal consequences of this Hypothesis:

**Theorem 1.3.** *If the Hypothesis 1.2 is true, then  $NP \subseteq NSPACE(\log^2 n)$ .*

*Proof.* We can simulate the computation  $M(w, u) = y$  in the Hypothesis 1.2 by a nondeterministic **square logarithmic** space Turing machine  $N$  in the computation of  $N(w)$ , since we can read the certificate string  $u$  within the read-once tape by a work tape in a nondeterministic **square logarithmic** space generation of symbols contained in  $u$  [8]. Certainly, we can simulate the reading of one symbol from the string  $u$  into the read-once tape just nondeterministically generating the same symbol in the work tapes using a **square logarithmic** space [8]. We could remove each symbol or a **square logarithmic** amount of symbols generated in the work tapes, when we try to generate the next symbol contiguous to the right on the string  $u$ . In this way, the generation will always be in **square logarithmic** space. This proves that  $L_2$  is in  $NSPACE(\log^2 n)$ . Due to  $L_2$  is closed under logarithmic space reductions in  $NP$ -complete, then every  $NP$  problem is logarithmic space reduced to  $L_2$ . This implies that  $NP \subseteq NSPACE(\log^2 n)$  since  $NSPACE(\log^2 n)$  is closed under logarithmic space reductions as well.  $\square$

## 1.2 The Problems

Now, we define the problems that we are going to use.

**Definition 1.4. SUBSET PRODUCT (SP)**

*INSTANCE:* A list of natural numbers  $B$  and a positive integer  $N$ .

*QUESTION:* Is there collection contained in  $B$  such that the product of all its elements is equal to  $N$ ?

*REMARKS:* We assume that every element of the list divides  $N$ . Besides, the prime factorization of every element in  $B$  and  $N$  is given as an additional data.  $SP \in NP$ -complete [5].

**Definition 1.5. Unary 0–1 Knapsack (UK)**

*INSTANCE:* A positive integer  $0^y$  and a sequence  $0^{y_1}, 0^{y_1}, \dots, 0^{y_n}$  of positive integers represented in unary.

*QUESTION:* Is there a sequence of 0–1 valued variables  $x_1, x_2, \dots, x_n$  such that

$$y = \sum_{i=1}^n x_i \cdot y_i?$$

*REMARKS:* We assume that the positive integer zero is represented by the fixed symbol  $0^0$ .  $UK \in NL$  [6].

## 2 Results

In number theory, the  $p$ -adic order of an integer  $n$  is the exponent of the highest power of the prime number  $p$  that divides  $n$ . It is denoted  $\nu_p(n)$ . Equivalently,  $\nu_p(n)$  is the exponent to which  $p$  appears in the prime factorization of  $n$ .

**Theorem 2.1.** *There is a deterministic **square logarithmic** space Turing machine  $M$ , where:*

$$SP = \{w : \exists u \text{ such that } M(w, u) = \text{“yes”}\}$$

when by  $M(w, u)$  we denote the computation of  $M$ ,  $w$  is placed on its input tape,  $u$  is placed on the special read-once tape of  $M$ , and  $u$  is polynomially bounded by  $w$ .

*Proof.* Given an instance  $(B, N)$  of  $SP$ , then for every prime factor  $p$  of  $N$  we could create the instance

$$0^y, 0^{y_1}, 0^{y_1}, \dots, 0^{y_n}$$

for  $UK$  such that  $B = [B_1, B_2, \dots, B_n]$  is a list of  $n$  natural numbers and  $\nu_p(N) = y, \nu_p(B_1) = y_1, \nu_p(B_2) = y_2, \dots, \nu_p(B_n) = y_n$  (Do not confuse  $n$  with  $N$ ). Under the assumption that  $N$  has  $k$  prime factors, then we can output in logarithmic space each instance for  $UK$  such that these instances of  $UK$  appears in ascending order according to the ascending natural sort of the respective  $k$  prime factors. That means that the first  $UK$  instance in

the output corresponds to the smallest prime factor of  $N$  and the last  $UK$  instance in the output would be defined by the greatest prime factor of  $N$ . Besides, in this logarithmic reduction we respect the order of the exponents according to the appearances of the  $n$  elements of  $B = [B_1, B_2, \dots, B_n]$  from left to right: i.e. every instance is written to the output tape as

$$0^z, 0^{z_1}, 0^{z_1}, \dots, 0^{z_n}$$

where  $\nu_q(N) = z, \nu_q(B_1) = z_1, \nu_q(B_2) = z_2, \dots, \nu_q(B_n) = z_n$  for every prime factor  $q$  of  $N$ . Finally, the certificate  $u$  would be a sequence of  $\{0, 1\}$  valued variables  $x_1, x_2, \dots, x_n$  such that for the first instance of  $UK$  we have

$$y = \sum_{i=1}^n x_i \cdot y_i,$$

for the second one

$$z = \sum_{i=1}^n x_i \cdot z_i,$$

and so on... We can simulate simultaneously  $k$  logarithmic space verifiers  $M_j$  for each  $j^{th}$  instance of  $UK$ . We can do this since the certificate  $u$  would be exactly the same for the  $k$  logarithmic space verifiers. Every logarithmic space verifier  $M_j$  uses  $O(\log |(B, N)|)$  space where  $|\dots|$  is the bit length function. So, we finally consume  $O(k \cdot \log |(B, N)|)$  space exactly in the whole computation of the **square logarithmic** space Turing machine  $M$ . We can assure that  $M$  verifies the instance  $(B, N)$  using the certificate  $u$  on **square logarithmic** space, because of  $k = O(\log N)$  and therefore, the whole computation can be made  $O(\log^2 |(B, N)|)$  space. To sum up, we can create this verifier that only uses a **square logarithmic** space in the work tapes such that the sequence of variables  $u$  is placed on the special read-once tape due to we can read at once every valued variable  $x_i$ . Hence, we only need to iterate from the variables of the sequence  $u$  from left to right to verify whether is an appropriated certificate according to the described constraints of the problem  $UK$  to finally accept the verification of all the  $k$  instances otherwise we can reject.  $\square$

**Theorem 2.2.**  $NP \subseteq NSPACE(\log^2 n)$ .

*Proof.* This is a directed consequence of Theorems 1.3 and 2.1 because of the Hypothesis 1.2 is true. Certainly,  $SP$  is closed under logarithmic space reductions in  $NP$ -complete. Indeed, we can reduced  $SAT$  to  $SP$  in logarithmic space and every  $NP$  problem could be logarithmic space reduced to  $SAT$  by the Cook's Theorem Algorithm [5]. Savitch's theorem states that for any space-constructible function  $S(n) \geq \log n$ , we obtain that  $NSPACE(S(n)) \subseteq DSPACE(S(n)^2)$  [9]. Based on this information, it is possible to directly determine that  $NSPACE(\log^2 n) \subseteq DSPACE(\log^4 n)$ .  $\square$

## Acknowledgments

The author thanks Emmanuel (CEO of NataSquad) for the financial support.

## References

- [1] Scott Aaronson.  $P \stackrel{?}{=} NP$ . *Electronic Colloquium on Computational Complexity, Report No. 4*, 2017.
- [2] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, USA, 2009.
- [3] Stephen Arthur Cook. The P versus NP Problem. <https://www.claymath.org/wp-content/uploads/2022/06/pvsnp.pdf>, June 2022. Clay Mathematics Institute. Accessed 9 January 2023.
- [4] Thomas Cormen, Charles Eric Leiserson, Ronald Linn Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, USA, 3rd edition, 2009.
- [5] Michael Randolph Garey and David Stifler Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W. H. Freeman and Company, USA, 1 edition, 1979.
- [6] Birgit Jenner. Knapsack problems for NL. *Information Processing Letters*, 54(3):169–174, 1995. doi:10.1016/0020-0190(95)00017-7.
- [7] Pascal Michel. A survey of space complexity. *Theoretical computer science*, 101(1):99–132, 1992. doi:10.1016/0304-3975(92)90151-5.
- [8] Christos Harilaos Papadimitriou. *Computational complexity*. Addison-Wesley, USA, 1994.
- [9] Walter John Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.
- [10] Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, USA, 2006.

NATASQUAD, 10 RUE DE LA PAIX 75002 PARIS, FRANCE <sup>\*1</sup>

**Email(s):** vega.frank@gmail.com <sup>\*1</sup> (corresponding author)