NP on Logarithmic Space

Frank Vega

August 30, 2023

NP on Logarithmic Space

Frank Vega *1

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 [8]. The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to this theoretical model for computation [8]. A deterministic Turing machine has only one next action for each step defined in its program or transition function [8]. 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 [8].

Let Σ be a finite alphabet with at least two elements, and let Σ^* be the set of finite strings over Σ [1]. A Turing machine M has an associated input alphabet Σ [1]. For each string w in Σ^* there is a computation associated with M on input w [1]. We say that M accepts w if this computation terminates in the accepting state, that is $M(w) =$ "yes" [1]. Note that, M fails to accept w either if this computation ends in the rejecting state, that is $M(w) =$ "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) [1].

P is the complexity class of languages that can be decided by deterministic Turing machines in polynomial time [2]. A verifier for a language L_1 is a deterministic Turing machine M, where:

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

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 [1]. 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 [6].

A function $f : \Sigma^* \to \Sigma^*$ 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 [8]. 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 $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$:

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

An important complexity class is *NP–complete* [3]. If L_1 is a language such that $L' \leq_l L_1$ for some $L' \in$ *NP–complete*, then L_1 is *NP–hard* [2]. Moreover, if $L_1 \in NP$, then $L_1 \in$ *NP–complete* [2]. The *NP–complete* class is formally defined by polynomial time reductions [3]. A principal *NP–complete* problem is SAT [3].

A logarithmic space Turing machine has a read-only input tape, a write-only output tape, and read/write work tapes [8]. The work tapes may contain at most $O(\log n)$ symbols [8]. In computational complexity theory, L is the complexity class containing those decision problems that can be decided by a deterministic logarithmic space Turing machine [6]. NL is the complexity class containing the decision problems that can be decided by a nondeterministic logarithmic space Turing machine [6].

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 [5]. 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 [5]. 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 [5].

We state the following Hypothesis:

Hypothesis 1.1. *There is an NP–complete language* $L_1 \in NSPACE(\log^2 n)$ *which is closed under logarithmic space reductions in NP–complete.*

We show the principal consequence of this Hypothesis:

Theorem 1.2. *If the Hypothesis 1.1 is true, then* $NP \subseteq NSPACE(\log^2 n)$.

Proof. Due to L_1 is closed under logarithmic space reductions in *NP–complete*, then every NP problem is logarithmic space reduced to L_1. This implies that $NP \subseteq NSPACE(\log^2 n)$ since $NSPACE(\log^2 n)$ is closed under logarithmic space reductions as well. □

1.1 The Problems

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

Definition 1.3. ***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 [3].*

Definition 1.4. ***Unary 0–1 Knapsack (UK)***

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

QUESTION: Is there a sequence of 0–1 valued variables $x_1, x_2, \ldots, x_n$ *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*.* $UK \in NL$ *[4].*

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. $SP \in NSPACE(\log^2 n)$.

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

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

for UK such that $B = [B_1, B_2, \ldots, B_n]$ is a list of n natural numbers and $\nu_p(N) = y, \nu_p(B_1) = y_1, \nu_p(B_2) = y_2, \ldots, \nu_p(B_n) = y_n$ (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 space reduction we respect the order of the exponents according to the appearances of the n elements of $B = [B_1, B_2, \ldots, B_n]$ from left to right: i.e. every instance is written to the output tape as

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

where $\nu_q(N) = z, \nu_q(B_1) = z_1, \nu_q(B_2) = z_2, \ldots, \nu_q(B_n) = z_n$ for every prime factor q of N. Finally, we generate a certificate on the work tapes that is a sequence of *0–1* valued variables $x_1, x_2, \ldots, x_n$ using **square logarithmic** space 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 a composition in logarithmic space reduction that simultaneously accept the k instances of UK. We can do this since the sequence certificate would be exactly the same for the k instances of UK. Every logarithmic space computation uses $O(\log \mid (B, N) \mid)$ space where $\mid \ldots \mid$ is the bit length function. So, we finally consume $O(k \cdot \log \mid (B, N) \mid)$ space exactly in the whole computation that would be **square logarithmic** space because of $k = O(\log N)$ and thus, the whole computation can be made $O(\log^2 \mid (B, N) \mid)$ space.

When we read one *0–1* valued variable x_i that is equal to 1 in the first instance of UK, then we store the current sum that includes adding the unary length of the element in the position i inside of the list. Next, we do the same for the remaining $k - 1$ instances of UK for the elements in the same position i. We store each current sum in the contiguous k instances of UK while we simultaneously copy these instances to the output tape from left to right. After that, we place the input head again in the first instance of UK and check whether the next *0–1* valued variable x_{i+1} is equal to 1 or not on the work tapes (We do not do nothing if the current *0–1* valued variable is equal to 0). We repeat over and over again this process without moving the output tape to the left during this composition of logarithmic space reduction [6]. In fact, we copy to the output tape the consecutive k instances of UK during this composition of logarithmic space reduction exactly the same number of times that the *0–1* valued variables in the certificate are equal to 1. Note that, the output tape of the inner Turing machine is the input tape of the outer Turing machine during this composition in logarithmic space reduction.

To sum up, we can create this composition in logarithmic space reduction that only uses a **square logarithmic** space in the work tapes such that the sequence of variables is placed on the work tapes due to we can read at once every valued variable x_i and remove it later. Hence, we only need to iterate from the variables of the sequence from the indexes 1 to n to verify whether we generate an appropriate certificate according to the described constraints of the problem UK to finally accept the k instances otherwise we can reject.

In addition, we can simulate the reading of one symbol from the string sequence of *0–1* valued variables into the work tapes just nondeterministically generating the symbol in the work tapes using a **square logarithmic** space [1]. 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 sequence of *0–1* valued variables. We could generate the certificate from the inner Turing machine in the composition of logarithmic space reduction and so, the outer Turing machine would be one-way deterministic during this composition of computations. In this way, the generation will always be in **square logarithmic** space. This proves that SP is in $NSPACE(\log^2 n)$. □

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

Proof. This is a directed consequence of Theorems 1.2 and 2.1 because of the Hypothesis 1.1 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 [3]. □

3 Conclusions

Savitch's theorem states that for any space-constructible function $S(n) \geq \log n$, we obtain that $NSPACE(S(n)) \subseteq DSPACE(S(n)^2)$ and therefore, $NSPACE(\log^2 n) \subseteq DSPACE(\log^4 n)$ [7]. Since $DSPACE(S(n))$ can be solved by a deterministic Turing machine in $O(2^{O(S(n))})$ time for any space-constructible function $S(n) \geq \log n$, then this would mean that $NP \subseteq QP$ (quasi-polynomial time class). We "believe" there must exist an evident proof of $NSPACE(\log^2 n) \subseteq P$ and thus, we would obtain that $P = NP$.

Acknowledgments

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

References

[1] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, USA, 2009.

[2] Thomas Cormen, Charles Eric Leiserson, Ronald Linn Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, USA, 3rd edition, 2009.

[3] 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.

[4] Birgit Jenner. Knapsack problems for NL. *Information Processing Letters*, 54(3):169–174, 1995. `doi:10.1016/0020-0190(95)00017-7`.

[5] Pascal Michel. A survey of space complexity. *Theoretical computer science*, 101(1):99–132, 1992. `doi:10.1016/0304-3975(92)90151-5`.

[6] Christos Harilaos Papadimitriou. *Computational complexity*. Addison-Wesley, USA, 1994.

[7] 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`.

[8] 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 *1
Email(s): vega.frank@gmail.com *1 (corresponding author)