NP on Logarithmic Space

Mediterranean International Conference of Pure & Applied Mathematics and Related Area (MICOPAM 2023)

Frank Vega, NataSquad

vega.frank@gmail.com

August 23-27, 2023

Abstract

The P versus NP problem 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 $DSPACE(S(n))$ and $NSPACE(S(n))$ for every space-constructible function $S(n)$. We prove that $NP \subseteq NSPACE(\log^2 n)$ just using logarithmic space reductions.

https://doi.org/10.33774/coe-2023-6n7v8-v2 **ORCID:** https://orcid.org/0000-0001-8210-4126 Content not peer-reviewed by Cam

Introduction

P versus *NP* is considered as one of the most important open problems in computer science.

The *P* versus *NP* problem belongs to the Smale's third problem on Steve Smale's list of eighteen unsolved problems.

This is one of the Clay Mathematics Institute's Millennium Prize Problems.

Definitions of P and NP

The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to the computational model (Arora and Barak, 2009).

In complexity theory, we describe the computational problems as languages where their elements are represented usually as binary strings (Arora and Barak, 2009). A complexity class is a set of languages putted together by some computational properties such as memory, execution time, etc (Arora and Barak, 2009).

P is the complexity class of languages that can be decided by deterministic Turing machines in polynomial time (Arora and Barak, 2009). *NP* is the complexity class of languages that can be decided by nondeterministic Turing machines in polynomial time (Arora and Barak, 2009).

Objectives

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, where $S(n)$ is a space-constructible function that maps the input size n to a non-negative integer (Arora and Barak, 2009).

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, where $S(n)$ is a space-constructible function that maps the input size n to a non-negative integer (Arora and Barak, 2009).

We prove that $NP \subseteq NSPACE(\log^2 n)$ just using logarithmic space reductions.

Methods

NP–complete problems are a set of problems to each of which any other *NP* problem can be reduced in polynomial time and they belong to *NP*. *SAT* is a well-known *NP–complete* problem.

We know that the complexity classes *NP* and $NSPACE(\log^2 n)$ are closed under logarithmic space reductions.

Proposition 1

If there is some NP–complete language L_1 which is closed under logarithmic space reductions in NP–complete and belongs to $NSPACE(\log^2 n)$, then

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

A certificate-based definition for *NL*

We can give a certificate-based definition for $NL = NSPACE(\log n)$ (Arora and Barak, 2009).

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 (Arora and Barak, 2009).

A certificate-based definition for *NL*

Definition 2

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 $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \{0,1\}^*$:

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

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 $|\ldots|$ is the bit-length function. The Turing machine M is called a logarithmic space verifier.

A Nondeterministic Problem on Time Complexity

Definition 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.
$SP \in NP\text{–}complete$ (Garey and Johnson, 1979).

A Nondeterministic Problem on Space Complexity

Definition 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. $UK \in NL$ (Jenner, 1995).

https://doi.org/10.33774/coe-2023-6n7v8-v2 **ORCID:** https://orcid.org/0000-0001-8210-4126 Content not peer-reviewed by Cam

Known Result

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.

Main Insight

This is the main insight.

Lemma 5

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

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 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.

https://doi.org/10.33774/coe-2023-6n7v8-v2 **ORCID:** https://orcid.org/0000-0001-8210-4126 Content not peer-reviewed by Cam

Finally, we generate a certificate 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 simultaneously k logarithmic space verifiers M_j for each j^{th} instance of UK.

We can done this since the sequence certificate would be exactly the same for the k logarithmic space verifiers.

Every logarithmic space verifiers M_j 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** 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 special read-once tape (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 reduction (Arora and Barak, 2009). In fact, we copy to the output tape the consecutive k instances of UK during this composition of logarithmic reduction exactly the same number of times that the *0–1* valued variables in the certificate are equal to 1.

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 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 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.

In addition, we can simulate the reading of one symbol from the string sequence of *0–1* valued variables into the read-once tape just nondeterministically generating the same symbol in the work tapes using a **square logarithmic** space (Arora and Barak, 2009).

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 reduction and so, the outer Turing machine would be deterministic during this composition of computations. In this way, the generation will always be in **square logarithmic** space. This proves that SP is in $NSPACE(\log^2 n)$. ■

Main Theorem

This is the main theorem.

Theorem 6

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

This is true since *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 (Garey and Johnson, 1979). ■

https://doi.org/10.33774/coe-2023-6n7v8-v2 **ORCID:** https://orcid.org/0000-0001-8210-4126 Content not peer-reviewed by Cam

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, $NSPACE(\log^2 n) \subseteq DSPACE(\log^4 n)$ (Savitch, 1970).

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 $P = NP$.

https://doi.org/10.33774/coe-2023-6n7v8-v2 **ORCID:** https://orcid.org/0000-0001-8210-4126 Content not peer-reviewed by Cam

References

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

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.

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

Walter John Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970. URL `https://doi.org/10.1016/S0022-0000(70)80006-X`.