DOI: 10.17516/1997-1397-2021-14-2-258-260
УДК 512.54

A Short Essay towards if P not equal NP

Vladimir V. Rybakov*
Siberian Federal University
Krasnoyarsk, Russian Federation
A. P. Ershov Institute of Informatics Systems
Novosibirsk, Russian Federation

Received 10.12.2020, received in revised form 10.01.2021, accepted 12.02.2020

Abstract. We find a computational algorithmic task and prove that it is solvable in polynomial time by a non-deterministic Turing machine and cannot be solved in polynomial time by any deterministic Turing machine. The point is that our task does not look as very canonical one and if it may be classified as computational problem in standard terms.

Keywords: deterministic computations, non-deterministic computations.

Citation: V.V. Rybakov, A Short Essay towards if P not equal NP, J. Sib. Fed. Univ. Math. Phys., 2021, 14(2), 258–260. DOI: 10.17516/1997-1397-2021-14-2-258-260.

1. Definitions and Formulations

To recall notation, P is the class of all computational tasks (problems) which may be solved by deterministic TMs in polynomial time, NP is the class of all computational tasks (problems) which may be solved by none-deterministic TMs in a polynomial time. Terminology and definitions concerning Turing Machines (TMs) and basics of mathematical theory of computation are supposed to be known for the reader.

What is a computational algorithmic task (problem)? Recall first what is a computable function? That is a function which may be computed by a TM. So, what is a solvable computable algorithmic task (Problem)?

That is a task which may be solved by a TM by an algorithm written in its program. That is — being passed by input data on the tape, TM works in accordance with its program and stops giving required output data (or just answer - yes - no -, if the task is a recognition of the input data).

We start by definition of the computational problem Pr. Consider the following computational task. It is a precise formal version of the Problem of Braking Coded Lock.

We model it by a Turing machine with the alphabet containing 0 and 1, an amount of marks for internal states $q_j, j \in J$, etc., as standard for a deterministic Turing machine etc. For any given set $(a_1, a_2, \ldots, a_n)$, where $a_i = 0$ or $a_i = 1$ we consider it as the code for our codded lock. We wish to open it. We describe below a TM solving this task.

(1) We put first $(a_1, a_2, \ldots, a_n)$ in the tape of TM and isolate in the final its part putting before first symbol a_1 a mark c showing that here we put the code. The machine cannot enter this part for any state before TM comes to comparison state. The tape cannot be extended after a_n. The part $a_1, a_2, \ldots, a_n$ cannot be edited.

(2) We then first put in the input of our TM any trial code $(c_1, c_2, \ldots, c_n)$, $(c_i = 0, 1)$ extending the tape to the left and compare it with $(a_1, a_2, \ldots, a_n)$. If we get total coincidence we put $(a_1, a_2, \ldots, a_n)$ outside and answer YES.

*Vladimir_Rybakov@mail.ru

(3) If $(c_1, c_2, \ldots, c_n)$ does not coinside with $(a_1, a_2, \ldots, a_n)$ at first met convergence we delete immediately from the tape of TM all $(c_1, c_2, \ldots, c_n)$ and only after this we get internal state of TM which only allow to move to the beginning of the tape.

(4) Then we start an early chosen, fixed and written in program of our TM algorithm A generating new trial code $(c_1, c_2, \ldots, c_n)$ and continue the comparison as in (2).

2. Results

We assume the algorithm A to be fixed, written in the instructions of our TM and might be applied for any given code $(a_1, a_2, \ldots, a_n)$. So our TM models both sides – codded lock owner, and code lock cracker.

Lemma 1. *Pr is a computational algorithmic computational task (problem) which is solvable in exponential time for some algorithm A for any given code.*

Proof. There are many known algorithms A doing this task in exponential time. The problem is algorithmic since we (may use) are using distinct algorithms A, computational as we use computations by TM.

It is time to emphasise here that the solution of the problem Pr consist of ONLY search (invention) and construction the algorithm A and writing it in the memory in commands of TM. Nothing else to do is prohibited. We cannot change the conditions, demands of the task. We are only allowed to find and construct algorithms A.

Lemma 2. *Pr is a computational algorithmic task (problem) which is solvable in polynomial time by a non-deterministic Turing machine.*

Proof is again evident. Just to take non-deterministic instructions in A while generating the input $I = \{c_1, \ldots, c_n\}$ (we need only n steps and $2n$ instructions to do it), and then we need only polynomial time to make comparison. □

All is fine, but the question is: where the proof that Pr cannot be solved in polynomial time with deterministic machine is? Here is the proof.

Lemma 3. *Pr cannot be solved in polynomial time by a non-deterministic Turing machine.*

Proof. Assume that there is a deterministic algorithm A solving this problem in polynomial time for any code $(a_1, \ldots, a_n)$ (working in accordance with the conditions (formulation) of the problem Pr as description above). Since the task does not allow to use the internal results of comparison trial inputs $(c_1, \ldots, c_n)$ with the code of the lock $(a_1, \ldots a_n)$, after rejecting the trial input $c_1, \ldots, c_n$ the machine TM again comes to generating new example $c_1, \ldots, c_n$, and then new one and so on, without any additional information about reason of rejecting - which symbols do not coincide.

The point here is that for any very initial trial input $(c_1, \ldots, c_n)$ the sequence of these new trial inputs $c_1, \ldots, c_n$ will be exactly the same as for any another given very first trial input (before finding (accepting) $(a_1, \ldots, a_n)$ and stopping). Why so? Because the algorithm A **is deterministic** — the sequence of steps and generated trial inputs $c_1, \ldots, c_n$ from A is predefined and does not depend on real value rejected trials $c_1, \ldots, c_n$, only it knows — if the trial does not coincide with the code $(a_1, \ldots, a_n)$ — we continue, generate a new one.

So, in any k-th step of applying A it generates at most k different trials $c_1, \ldots, c_n$. Let then $a_1, \ldots, a_n$ be the tuple which does not belong to trials which may be generated by A in its $2^n - 1$ consequent applications. Then if $a_1, \ldots, a_n$ is taken as the code our TM will require at least 2^n steps before crack $a_1, \ldots, a_n$. □

Thus as the result we have

Theorem 4. *$PR \in NP$ and $PR \notin P$, so $P \neq NP$.*

Clearly that the interest to non-deterministic computation comes (primarily?) from famous Cook-Levin theorem, cf. [1]. There is a big amount of papers towards if $P = NP$ or $P \neq NP$, cf. [1–4].

We do not touch in this short notice complexity theory and the problem satisfiability in Boolean logic and hence the famous Cook-Levin theorem. Just we would like to emphasize that if not to fix very precisely the meaning what is a computational algorithmic problem — it may confuse the researchers and put efforts aside. So, if we use general — and very plausible and even rather convincing interpretation — as in this paper — we shortly obtain the negative answer on if $P = NP$. Though questions about behaviour of Turing machines and how algorithms work on them in accordance with their programs and precisely specified tasks are definitely computational algorithmic problems (maybe internal ones — but nonetheless). Therefore the question of what is a computational algorithmic problem definitely needs clarification.

This research is supported by High Schools of Economics (HSE) Moscow; supported by the Krasnoyarsk Mathematical Center and financed by the Ministry of Science and Higher Education of the Russian Federation (Grant No. 075-02-2020-1534/1).

References

[1] S.A.Cook, The complexity of theorem proving procedures, Proceedings of the Third Annual ACM Symposium on Theory of Computing, 1971, 151–158. DOI: 10.1145/800157.805047.

[2] https://en.wikipedia.org/wiki/P_versus_NP_problem

[3] Javier A.Arroyo-Figueroa, The Existence of the Tau One-Way Functions Class as a Proof that P != NP, 2016. https://arxiv.org/abs/1604.03758

[4] Mathias Hauptmann, On Alternation and the Union Theorem, Mathematics, Computer Science, 2016. https://arxiv.org/abs/1602.04781

Заметка о проблеме равенства P и NP

Владимир В. Рыбаков
Сибирский федеральный университет
Красноярск, Российская Федерация
Институт систем информатики им. А. П. Ершова
Новосибирск, Российская Федерация

Аннотация. В статье вводится алгоритмическая проблема и доказывается, что она разрешима за полиномиальное время на недетерминированных машинах Тьюринга и не решается за полиномиальное время на детерминированных машинах Тьюринга. В то же время, введенная проблема не выглядит как стандартная в общепринятом понимании и не самоочевидно может ли она быть классифицирована как каноническая.

Ключевые слова: детерминированные вычисления, недетерминированные вычисления.