The Cook-Levin Theorem is False

André Luiz Barbosa
http://www.andrebarbosa.eti.br
Non-commercial projects: SimuPLC – PLC Simulator & LCE – Electric Commands Language

Abstract. *This paper demonstrates that the Cook-Levin Theorem is false, so the NP-Completeness, since a faulty hidden assumption is tacitly used in its proofs. I am so sorry.*

Mathematics Subject Classification (2010). Primary 68Q15; Secondary 68Q17.

Keywords. Computational Complexity, Cook-Levin Theorem, Cook-Levin-Barbosa Theorem, NP/NP_g-Completeness, Observer-Dependency Complexity (ODC).

Contents

1. Introduction

The Cook-Levin Theorem (Cook's Theorem) [7] is the most central theorem in the Computational Complexity Theory, stating the exceptionally important concept of the NP-Completeness, but very very unfortunately it is false, since a faulty hidden assumption is tacitly used in its proofs. The researchers from this area never thought about that hidden assumption and it was so lamentably always neglected for the last four decades. Why?

In this little paper, I present a theory in order to explain what occurred and why this mistake has such a so long term. About this matter, see the comments in [8] and [13].

2. The falseness of the Cook-Levin Theorem

Looking in [1] more general definitions for decision problems and polynomial-time

DTM (Deterministic Turing Machine), and seeing deeper the concerned old traditional definitions, an extremely surprising fact – that will bemuse the Theoretical Science community – is discovered: The central "theorem" from the Computational Complexity Theory crumbles to the ground, so the NP-Completeness (even though we refuse to see…):

Theorem 2.1. The Cook-Levin Theorem (CLT: SAT is NP-Complete) is false (SAT is NOT NP-Complete: SAT $\in$ P does NOT imply P = NP).

Proof. The Cook-Levin Theorem is false, since in order to all its proofs work a fundamental – but in general neglected – input (information) is absolutely essential and tacitly supposed to be known and given to us (or can be computed in deterministic poly-time) in their scope [2]: the polynomial running time *p(n)* of some NTM (Nondeterministic Turing Machine) that decides an NP problem (n = |input for it|). With such a *p(n)* anyway provided (which can be considered a hidden assumption or axiom), a poly-time DTM that reduces any instance of this problem to an instance of poly-size Boolean formula (or of poly-size Boolean circuit that simulates the problem) can always be constructed by us within deterministic poly-time (where the size of this generated Boolean formula (or circuit) is in $O(log(n)\, p(n)^2)$).

However, if such a *p(n)* is unknown (or is not given to us) neither can be computed in deterministic polynomial time, then all these proofs shall fail. They stop utterly working, consequently, when such a *p(n)* is unknown (or not given to us) *a priori*, since it cannot be computed in deterministic polynomial time, as proved in the Proposition below:

Proposition 2.1. Computing in deterministic polynomial time a polynomial $\mathbf{n^k}$ (where **n** = |input|) that upper bounds the running time of an arbitrary polynomial-time DTM (therefore, NTM) M is impossible. (Hence, computing in deterministic polynomial time that *p(n)* – information that is essential in order to CLT works – is impossible too).

Proof. Assume that there is a polynomial-time DTM B that returns **k** on the description of any arbitrary DTM A promised to be polynomial-time, where the polynomial $\mathbf{n^k}$ *must* upper bound the running time of A. Then, let A be the polynomial-time DTM (computer program with unbounded memory) below:

```
01. A(input) {
02.   n := length(input);
03.   k := B(⟨A⟩); // the DTM B runs on the proper description (or code) of A and returns k
04.   execute n^(k+1) arbitrary steps; // this diagonalization makes the DTM B always wrong
05.   if (k < 8) return(0); else return(1);
06. } // note that we can use the proper description of A here by the Recursion Theorem [2]
```

Thus, as it is easily seen above, the diagonalization in the lines 03 and 04 makes that the running time of the DTM A is at least $\Theta(n^{k+1})$, whereas the DTM B answers that this running time is only $O(n^k)$. Hence, cannot exist at all a polynomial-time DTM that on the description of an arbitrary polynomial-time DTM computes a finite **k** where $\mathbf{n^k}$ upper bounds its polynomial running time: The CLT is false. So, the NP-Completeness is really incomplete, a sweet but evanescent delusion. □

Obs.: I know that the proof of the CLT does not rely on the negating the Proposition 2.1 (neither on the correctness nor incorrectness of the algorithm above), but if it was false, then the arguments in this paper would not work in order to refuse the CLT.

In short: The CLT is really a kind of *constructive proof* [16] and algorithmic process that allows actually constructing a real and legitimate polynomial-time computable function

CLT': $\Sigma^* \times \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$, where if CLT'($\langle M \rangle$, **x**, **p**) = **b**, then **p** is a polynomial in |x| (that *p(n)* with n = |x|), and **b** is a poly-size ($O(log(n)\, p(n)^2)$) Boolean formula that represents the entire computation of *M* on input **x**, where **b** is satisfiable if and only if the polynomial-time NTN *M* (that decides an NP language *L*) accepts **x** within **p**(|x|) computational steps (that is, $\mathbf{b} \in SAT \Leftrightarrow \mathbf{x} \in L$, which implies that if SAT should be in P, then P = NP).

However, note that in order to really construct **b** (as described above) $\langle M \rangle$, **x** and **p** MUST be given. If that *p(n)* of *M* is not given, then effectively constructing that Boolean formula **b** by means of the CLT process above is impossible within polynomial time, of course. So, the existence of a polynomial-time DTM that decides the SAT does NOT imply that it can always be constructed another polynomial-time DTM that decides a given NP language, undoubtedly, taking into account the [our] impossibility of the concrete poly-time construction of the Boolean formula **b** above when *p(n)* is unknown or not given to us.

See that the input arguments $\langle M \rangle$ and **x** for CLT' can always be given, of course, but if **p** cannot be it, then we can use in its place into that function any arbitrary polynomial **q**, obviously. But this does not work, in general, since that involved algorithmic process upon CLT will compute **q**(|x|) in order to know what the size of the tableau it must construct in order to build that Boolean formula **b**. [2] Since this formula is satisfiable if and only if *M* accepts **x** within **q**(|x|) computational steps, if $\mathbf{q}(|x|) < \mathbf{p}(|x|)$, then this formula will not represent the entire computation of *M* on input **x** when *M* uses more than **q**(|x|) steps in order to accept **x**, clearly (because *M* cannot use more than **p**(|x|) steps [for $T_M(|x|) = \mathbf{p}(|x|) = p(n)$, where $n = |x|$], but, in this case, *M* can use more than **q**(|x|) ones: hence, in this instance, *M* accepts **x**, but **b** will not be satisfiable, which destroys the correctness of this reduction in general, as that *p(n)* is unknown or not given to us).

Consequently, the CLT states if SAT is in P, then ALL NP problems (that can always be decided by some polynomial-time NTM) are in P too, but the big issue, that destroys the CLT and its beautiful and marvelous conclusion, is that that polynomial *p(n)* (the input argument **p** for CLT') can be unknown or not given to us (remember that it is not computable in deterministic polynomial time, by Proposition 2.1) and even then that *L* is an NP language, of course, which implies that there are NP languages that cannot be reduced to SAT in deterministic polynomial time, which – in theoretical terms – annihilates completely the CLT and the proper concept of NP-Completeness, since they are intimately and utterly linked.

In fact, the class NP can be divided into two new disjoint classes: NP_g (when that *p(n)* is known and given) and NP_u (when *p(n)* is unknown or not given to us), where $NP = NP_g \cup NP_u$, and $NP_k \cap NP_u = \varnothing$. Into traditional beliefs, NP is considerate equal to NP_g, and NP_u is considerate equal to $\varnothing$ (the hidden assumption treated here), but these considerations take not account that the class NP_u can be a genuine, useful and very important complexity class into the development of the Computational Complexity Theory, with great powerful applications in mathematically proven polynomial-time-unbreakable public-key cryptography, for instance (I am writing a paper about this practical application of the NP_u). By the way, see [18].

Into more formal terms, lets see the definitions for the two new disjoint classes that build the traditional class **NP**: $\mathbf{NP_g}$ and $\mathbf{NP_u}$ (Nondeterministic Polynomial Time when the involved polynomial time is or not given to us, respectively):

Definition 2.1. $\mathbf{NP_g}$**.** Let L be a language over Σ. $L \in \mathbf{NP_g}$ if and only if there is a binary relation $R \subseteq \Sigma^* \times \Sigma^*$ and a known and given finite fixed positive integer ***p*** such that the following two conditions are satisfied (note that here ***p*** must be fully known for all involved people, that must be fully conscious of this fact):

1. For all $x \in \Sigma^*$, $x \in L \Leftrightarrow \exists y \in \Sigma^*$ such that $(x, y) \in R$ and $|y| \in O(|x|^p)$; and
2. The language $L_r = \{x\#y : (x, y) \in R\}$ over $\Sigma \cup \{\#\}$ is decidable by a polynomial-time DTM whose polynomial is known and given to us. □

Definition 2.2. NP_u. Let L be a language over Σ. $L \in \mathbf{NP_u}$ if and only if there is a binary relation $R \subseteq \Sigma^* \times \Sigma^*$ and a known and given finite fixed positive integer $\boldsymbol{p}$ such that the following two conditions are satisfied (where $\boldsymbol{p}$ must obey the same clause as on Def. 2.1):

1. For all $x \in \Sigma^*$, $x \in L \Leftrightarrow \exists y \in \Sigma^*$ such that $(x, y) \in R$ and $|y| \in O(|x|^p)$; and
2. The language $L_r = \{x\#y : (x, y) \in R\}$ over $\Sigma \cup \{\#\}$ is decidable by a polynomial-time DTM whose polynomial is unknown or not given to us. □

Definition 2.3. NP. $NP = NP_g \cup NP_u$. □

In the definitions above, a DTM that decides L_r is called a *verifier* for L and a **y** such that $(x, y) \in R$ is called a *certificate of membership* or *witness* of **x** in L.

Observer-Dependency Complexity (ODC): *Knowing* means here that a person knows completely that involved polynomial and is fully conscious of this fact; hence the definition of NP_g is surely *subjective*, in the sense that a problem may be in NP_g [NP_u] for a person but not for another one. Questions such as "Is L in NP_g [NP_u]?" can have different answers for different people. Although whether it is in NP continues an objective matter. Notice that here *we* are the *Observer* on the ODC concept, being fully conscious of this fact.

Thus, as with Quantum Mechanics in respect to Physics, this revolutionary and seminal concept (ODC) puts the *observer* into center of TCS stage. The human factor eventually wins again, even into the abstract aridity of Computational Complexity Theory: Turing Machine and other similar formalisms are very important and powerful intellectual tools, but we, people that non-mechanically think ["– *cogito, ergo sum.*"], are more than it!

See that NP_u languages are legitimate NP ones, but they cannot be reduced to SAT in deterministic polynomial time, as proved over, which implies that the CLT is false, plainly. □

2.1 Example of an NP problem not reducible to SAT by means of the CLT

Definition 2.4. Polynomial Acceptance M_f problem (PA-M_f). The **PA-M_f** is defined over {0, 1} where $\mathbf{w} \in$ **PA-M_f** if and only if $\mathbf{w} = \langle \mathbf{1}^{\mathbf{n}} \rangle$ and the fixed polynomial-time DTM $\boldsymbol{M_f}$ (whose polynomial is unknown or not given to us) on some (at least one) input **x** of length **n** writes **1** on the cell 0 of the tape and then halts. The DTM $\boldsymbol{M_f}$ allows as input any **n**-bit word and is promised to eventually always write **0** or **1** on the cell 0 and then halt, within polynomial time: $T(n) = O(n^k)$, for some fixed nonnegative finite constant **k**, which is unknown or not given to us. □

Notice that the fixed DTM $\boldsymbol{M_f}$ is specifically built and given to this problem, and it can be any polynomial-time DTM (or computer program with unbounded memory) promised to have such a behavior above.

See that the **PA-M_f** is in NP (more exactly, in NP_u with respect to us), evidently, since if $\mathbf{w} \in$ **PA-M_f**, then it is verifiable in deterministic polynomial time whether the DTM $\boldsymbol{M_f}$ writes **1** on the cell 0 and then halts on an [guessed] input **x** of length **n**: it suffices to simulate the running of the DTM $\boldsymbol{M_f}$ on that input and then to check the contents of that cell 0 after the

simulation has finished. Note that simulating the running of any polynomial-time DTM on an arbitrary input can be done within polynomial time too (since a polynomial is a time-constructible function [14]).

But the **PA-M_f** is not reducible to SAT by means of the CLT, noticeably, because the polynomial time running of the DTM M_f is unknown or not given to us, hence a poly-sized Boolean formula (or circuit) that represents that instance of the **PA-M_f** to be decided cannot be really [concrete and effectively] built by us in deterministic polynomial time: So, the CLT is here concretely proven false.

2.2 How can a Theorem be false?

Since a *theorem* is an absolute mathematical truth, how can the CLT be false?

– I am so sorry, but the answer is unavoidable: This "theorem", in fact, is not a genuine theorem, for its proofs are ill, since into them it is tacitly used the hidden non-proved (faulty, essentially) assumption (or hidden axiom, contradictory with ZFC [4]) – by Proposition 2.1 –, that states that a polynomial *p(n)* that upper bounds the running time of some NTM that decides in nondeterministic polynomial time the problem to be reduced into that reduction to the SAT can be always anyway provided (either that *p(n)* is known and given *a priori* or can be computed by a polynomial-time DTM). We have seen above (Proposition 2.1) that this is not always accurate, obviously.

Another likely source to the faulty "proofs" could yet be a hidden assumption into the definition of polynomial-time NTM, veiling defining that related *p(n)* as always *a priori* known and given to us. However, in this definition there is not such assumption. [3]

This hidden assumption was a big flaw into all the known "proofs" of the CLT, unknown for four decades, which demonstrates how much the Theoretical Computer Science is an extraordinary marvelous land, and, at same time, a very dangerous math territory.

In order to illustrate the danger, suppose that A has written a polynomial-time computer program (polynomial-time DTM) that correctly decides the SAT. Then, B gives to A a problem *Y* that is proven to be decidable within $\mathbf{n^k}$ nondeterministic steps (hence, *Y* is an NP problem), and then solicits to A another polynomial-time computer program that correctly decides the problem *Y* (the CLT assures absolutely that, given a polynomial-time DTM that decides the SAT, this required program can be concretely written).

Then A, on the other hand, asks B what is the value of **k**. B answers to A that it is unknown: The proof that *Y* is in NP does not determine **k** at all, even though it states that **k** is *some* fixed finite nonnegative constant. Thus, A responds to B that the demanded task is impossible without knowing **k**. Whence, as to know *a priori* or compute **k** in deterministic polynomial time is in general unfeasible (by Proposition 2.1, for instance), B (the entire World, really) concludes that the CLT is false, so the NP-Completeness, miserably.

Why did the proofs of the Cook-Levin Theorem be considered true?

– Because that polynomial *p(n)* was supposed to be *a priori* a *fully known and given* part of *all* NP problems, which this paper has proved to be in general a false statement, with the **PA-M_f** in Section 2.1.

But, why was that polynomial supposed *a priori* to be always a fully known and given information?

– Because all the actual decision problems that motivated the initial studies on Computational Complexity Theory are decidable by poly-time DTM or NTM whose involved polynomials are *a priori* always fully known and given to us, e.g., SAT, Graph Connectivity, Primality Testing, Matrix Determinant, LP, Hamilton Cycle, Steiner Tree, Graph 3-Coloring, Max-Clique. So, all they are very different from my **PA-M_f** problem, as we have seen here.

3. Fixing the Cook-Levin Theorem: The Cook-Levin-Barbosa Theorem

Theorem 3.1. NP_g-Completeness. The Cook-Levin-Barbosa Theorem (CLBT). SAT $\in$ P iff P = NP_g (SAT is NP_g-Complete).

Proof. Any traditional standard proof of the old Cook-Levin Theorem goes here, since into class NP_g that polynomial *p(n)* IS known and given to us, by Def. 2.1. □

Hence, although SAT is not NP-Complete, it is in fact NP_g-Complete, by the CLBT!

4. Conclusion

The conclusion is that several theorems into TCS should be reviewed, searching by possible faulty hidden assumptions tacitly used into their proofs, which can preclude the correctness or the soundness of the involved mathematical truths.

The Savitch's Theorem, [5] for instance, another central theorem in the Computational Complexity Theory, is also false, since it tacitly uses similar faulty hidden assumption too: that a polynomial *s(n)* that upper bounds the running space of some NTM that decides in nondeterministic polynomial space the problem can be always anyway provided for us (either that *s(n)* is known and given *a priori* or can be computed by a polynomial-space DTM), which is also false, by analogous method used here. See that these hidden assumptions hurt even all the Polynomial Hierarchy, [10] and other similar constructions – though all of these ones can be generalized and fixed by the Barbosa's Program, as described in [1].

So, it is here finished this extraordinary eleven-page paper that will change the four-decade TCS and alter almost everything fundamental in such an area!

5. The Reviews Battles

Suppose a reviewer says that:

"– I do not understand the theoretical model where a problem *Q* is solved by an *f*(n)-bounded NTM but *f*(n) is not known."

That one has unfortunately misunderstood the ideas that uphold the paper, noticeably: In our *f*(n)-bounded NTM, we know that this *f*(n) is always *some* polynomial *p(n)*, and we are fully conscious of this fact, although perhaps we do not know what *exactly* that *p(n)* is.

Suppose another reviewer says that:

"– The CLT states that for each NP machine *C* there is a polynomial-time DTM M_C that reduces the language accepted by *C* to SAT and depends on *C*, where the polynomial

bound of C is [anyway, maybe miraculously] hardwired in the definition of M_C, and the translation from C to M_C is NOT required to be effective, only that an M_C exists for each C is required. By this, in order to the reduction works, all that would be required there exists a polynomial $p(n)$ that bounds the running time of C. Hence, the claims of the author are clearly incorrect, since in order to prove the CLT, we do not need to specify an exact value of $p(n)$, but we need only to know that such a $p(n)$ exists, as in the definition below:

Definition 5.1. NP-Completeness. Let L be a language over a finite alphabet Σ. L is **NP-Complete** if and only if the following two conditions are satisfied:

1. $L \in$ **NP**; and

2. Any language C in **NP** is deterministic-polynomial-time-reducible to L (written as $\boldsymbol{C \leq_p L}$), where $\boldsymbol{C \leq_p L}$ if and only if the following two conditions are satisfied:

   2.1 There exists $f : \Sigma^* \rightarrow \Sigma^*$ such that for all w in Σ^*, $w \in C \Leftrightarrow f(w) \in L$; and

   2.2 There exists a polynomial-time DTM M that halts with $f(w)$ on its tape on any input w [although this M maybe **cannot** be really built]. □"

So, he/she continues, I would be wrong and I should read the basic definitions and the rudiments of theory of NP-Completeness, in order to I see my huge error.

I have followed that reviewer advice, but the literature about the matter (like in [9]) and standard texts on Theory of Computation (like in [2]) affirm contrarily that the CLT states that: A problem Z in NP is also in NP-Complete (NPC) if and only if EVERY other problem in NP can be **transformed** into Z **in polynomial time**. In [11] it is even written, about NP-Completeness: "It is not known whether any problem in NPC is tractable [decidable in polynomial time], but an important property of this class is that every problem in the class is tractable if and only if one such problem is tractable." More clear impossible.

Hence, the definition of NP-Completeness above, relying only on the existence criterion of the DTM M (without its actual construction), is entirely faulty and useless, since by it the existence of a deterministic polynomial-time algorithm that decides an NP-Complete problem does not imply that every NP problem can be decided in polynomial time, despondently, that is, SAT in P would not lead to [even the traditional question] NP = P.

So, for our happiness' sake, see a correct definition:

Definition 5.2. NP-Completeness. Let L be a language over a finite alphabet Σ. L is **NP-Complete** if and only if the following two conditions are satisfied:

1. $L \in$ **NP**; and

2. Any language C [over a finite alphabet Σ' (maybe different from Σ)] in **NP** is deterministic-polynomial-time-reducible to L (written as $\boldsymbol{C \leq_p L}$), where $\boldsymbol{C \leq_p L}$ if and only if the following two conditions are satisfied:

   2.1 There exists $f : \Sigma'^* \rightarrow \Sigma^*$ such that for all w in Σ'^*, $w \in C \Leftrightarrow f(w) \in L$; and

   2.2 There exists a polynomial-time DTM M that halts with $f(w)$ on its tape on any input w, and this M can **always** be really built. □

"– Proving a problem in NP to be NP-complete tells us that it is as hard to solve as

any other NP problem. Said another way, if there is any NP-complete problem that admits an efficient solution, then every NP problem does so." is in [12]. By the way we have seen above, more clearly impossible, yet.

Thus, in order to the CLT works, we need more than only to know that that polynomial *p(n)* exists: We need definitely to know exactly what this polynomial is (8000?, a googol?, 3n?, $5n^2 + 7n$?, …?, $11n^{100} + 13n^{10} + 17n^8$?, …?), naturally.

Hence, by the CLT, if any problem in NPC is in P, then all the problems in NP are [concrete and effectively, obviously] in P too, because the definition of NP-Completeness into CLT. Then, as demonstrated in Section 2 (there are actual NP languages that cannot be reduced to SAT in deterministic polynomial time, like the **PA-M$_f$** problem, by Def. 2.4), by its proper pretensions, the CLT is not true at all, of course.

Finally, yet another reviewer has said that:

"– I can do no more than to emphasize that regardless of any ambiguous phrasing the author may find in Wikipedia or textbook accounts that he believes supports his mistaken view, published papers and reputable researchers in the field rely on the correct definition: *A problem* C *is NP-Complete if it is in NP and every problem* A *in NP reduces to* C." Period.

Sadly, that reviewer has forgot the main point of the really correct definition: *A problem* C *is NP-Complete if it is in NP and every problem* A *in NP reduces to* C ***in polynomial time***. Otherwise, the NP-Completeness would be useless even to the most reputable researcher in the field from the world, since without this time-restriction any NP problem *C* (except the trivial ones: Σ^* and Ø) would be NP-Complete too: it would be sufficient to reduce any problem *A* in NP to *C* in exponential time, which is always possible by means of a *brute-force* machine, of course.

In [15] it is even defined: "**Definition 15.3.** We say that a recognition problem A_1 *polynomially transforms* to another recognition problem A_2 if, given any string *x*, **we can construct** a string *y* within polynomial (in |*x*|) time such that *x* is a *yes* instance of A_1 if and only if *y* is a *yes* instance of A_2." and "**Definition 15.4.** A recognition problem A $\in$ *NP* is said to be *NP-complete* if all other problems in *NP* polynomially transform to A."

Notice that excerpt that I have stressed "**we can construct**": Papadimitriou and Steiglitz agree with me that the definition of NP-Completeness cannot rely only on the existence criterion of the poly-time construction of that string *y*, but otherwise on its actual construction within polynomial time, which is not possible for the **PA-M$_f$** problem (Def. 2.4), for instance.

Thus, the huge issue with the CLT, hence, is that that real reduction cannot work within polynomial time if the running time of some NTM that decides *A* is not exactly given, which implies that the CLT is false, as proven here.

In fact, I think the reviewers of this paper should, at least, ask themselves (with respect to CLT proofing and Defs. 2.1, 2.2, 2.3, 2.4, 5.1 and 5.2) and then respond on their reviews:

1. Does there be really a hidden assumption in the CLT proofs? Why or why not?
2. Are the CLT proofs, in fact, *non-constructive ones* [16]? Why or why not?
3. Does **NP = NP$_g$**? Why or why not?
4. Does **NP$_u$ = Ø**? Why or why not?
5. Is the **PA-M$_f$** problem in NP, by Def. 2.4, and **PA-M$_f$ $\leq_p$ SAT**? Why or why not?

6. Is the Def. 5.1 better than the Def. 5.2? Why or why not?
7. Is the *Observer-Dependency Complexity* a relevant concept? Why or why not?
8. If this paper had been perfectly written, with great English usage without seeming arrogant and with more "*respectful*" (flattering) style, would your answers to questions above be different? Could a supposed author's antipathy transform his correct mathematical ideas into wrong ones? Is this fair? Why or why not?

Finally, I beg you to ask yourself and then respond primarily to the key question that can redefine your entire field of research and knowledge:

– Does the discovery (or invention/creation) of a deterministic poly-time SAT-decider algorithm imply all NP problems can be actually decided in deterministic polynomial time, even those ones in NP_u? Does your answer here contradict the previous one to question 6 above, on the value of the definitions 5.1 and 5.2? Why or why not? (I beg you to think very profoundly on this key question – a really metamathematical and philosophical issue.)

6. Pride & Prejudice

Opinions from a reviewer:

Pride: "– It can be easily seen that the paper is wrong, and I will briefly detail why this is the case: We do not actually get to know P-time algorithms for all problems in NP when someone would give a P-time algorithm for SAT. We, as theoreticians, are (in this case) happy enough with existence of algorithms. If someone were to give a P-time algorithm for some NP-hard problem then we might not really be much closer solve our favorite NP-complete problems. Indeed, we only know that algorithms exist and we also do not know their runtimes."

Prejudice: "– The author is free to contest our way of making theory about efficient algorithms, in the sense that they do not capture his view of the real world. There is no problem with that. But I would kindly ask him to stop sending his opinions to theory conferences or journals. The established theory of NP-completeness, and other notions, are the language that we speak and love, and it is easy even for students to understand where this present paper breaks down, and why the Cook-Levin-Theorem does after all hold."

Pride and Prejudice: "– NP-completeness is not perfect, in that it is often inconceivably hard to capture what practical instances of problems are (and often their are much easier than the worst case). However, this does not harm the fact that its fundamental results (like the Cook-Levin-Theorem) are correct, even if they do not match the author's intuition."

7. What would the proper Stephen Cook say?

"– It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be *reduced* to the problem of determining whether a given propositional formula is a tautology. Here *reduced* means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second." are the first phrases that Stephen Cook wrote on his seminal 1971 paper *The Complexity of Theorem-Proving Procedures* [17], introducing the first proof of his main theorem.

What would he mean by the stressed stretch "can be solved"? I think that he means, "can be solved", but the reviewers of this my paper seem do not believe him, so far, guessing that he would be saying something like "there is (or must be) a poly-time DTM (although maybe nobody can construct it at all) that solves it". So, I would recommend that these reviewers turn on to believe more on what the great masters wrote and said into their great masterpieces.

Continuing with that Cook's masterpiece:

"**Theorem 1.** *If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to {DNF tautologies}.*

Corollary. *Each of the sets in definitions 1)–5) is P-reducible to {DNF tautologies}.*

This is because each set, or its complement, is accepted in polynomial time by some nondeterministic Turing machine.

Proof of the theorem. Suppose a nondeterministic Turing machine *M* accepts a set *S* of strings within time *Q(n)*, where *Q(n)* is a polynomial. Given an input *w* for *M*, we will construct a proposition formula *A(w)* in conjunctive normal form such that *A(w)* is satisfiable iff *M* accepts *w*. …" [17]

Once again, what would Stephen Cook mean by the stressed stretch "we will construct"? I think that he means, "we will construct", but so far the reviewers of this my paper seem definitely do not believe him.

8. Freedom & Mathematics

"**– The essence of Mathematics is Freedom.**" (Georg Cantor) [6]

9. References

[1] A. L. Barbosa, *P != NP Proof*, unpublished, available: http://arxiv.org/ftp/arxiv/papers/0907/0907.3965.pdf

[2] M. Sipser, *Introduction to the Theory of Computation – Second Edition*, Thomson Course Technology, Boston MA, 2006.

[3] From Wikipedia, the free encyclopedia, "*P versus NP Problem*", unpublished, available: http://en.wikipedia.org/wiki/P_versus_NP_problem

[4] From Wikipedia, the free encyclopedia, "*Zermelo-Fraenkel Set Theory*", unpublished, available: http://en.wikipedia.org/wiki/Zermelo-Fraenkel_set_theory

[5] From Computational Complexity, blog, "*Foundations of Complexity – Lesson 18: Savitch's Theorem*", posted at May 14, 2003, by L. Fortnow, unpublished, available: http://blog.computationalcomplexity.org/2003/05/foundations-of-complexity-lesson-18.html

[6] From The Engines of Our Ingenuity, site, "*Episode nº 1484: GEORG CANTOR*", posted

by John H. Lienhard, unpublished, available: http://www.uh.edu/engines/epi1484.htm

[7] M. Johnson, *Handout* "*Lecture 5: The Cook-Levin Theorem*", unpublished, available: http://russell.lums.edu.pk/~archive/Complexity/durham/lecture5handout.pdf

[8] From Gödel's Lost Letter and P=NP, a personal view of the theory of computation, blog, public comments on "*Facts No One Really Checks*", posted at July 25, 2012, by R. J. Lipton, unpublished, available: http://rjlipton.wordpress.com/2012/07/25/facts-no-one-really-checks/#comment-22187

[9] From Wikipedia, the free encyclopedia, "*NP-Complete*", unpublished, available: http://en.wikipedia.org/wiki/NP-complete

[10] From Wikipedia, the free encyclopedia, "*Polynomial Hierarchy*", unpublished, available: http://en.wikipedia.org/wiki/Polynomial_hierarchy

[11] Michael P. Drazin, book review of *Computers and intractability: A guide to the theory of NP-completeness, by Michael R. Garey and David S. Johnson, W. H. Freeman and Company, San Francisco, 1979.* Bulletin (New Series) of the American Mathematical Society, Volume 3, Number 2, September 1980, pp. 898-904, available: http://www.ams.org/journals/bull/1980-03-02/S0273-0979-1980-14848-X/S0273-0979-1980-14848-X.pdf

[12] L. Fortnow and S. Homer, *A Short History of Computational Complexity*, unpublished, available: http://people.cs.uchicago.edu/~fortnow/papers/history.pdf

[13] From Computational Complexity – Computational Complexity and other fun stuff in math and computer science from Lance Fortnow and Bill Gasarch, blog, public comments on "*Who do you write papers for?*", posted at January 31, 2013, by Lance Fortnow, unpublished, available: http://blog.computationalcomplexity.org/2013/01/who-do-you-write-papers-for.html?showComment=1359738809162#c883623509379550213

[14] From Wikipedia, the free encyclopedia, "*Constructible Function*", unpublished, available: http://en.wikipedia.org/wiki/Constructible_function

[15] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications, Inc., Mineola NY, 1998.

[16] From Wikipedia, the free encyclopedia, "*Constructive Proof*", unpublished, available: http://en.wikipedia.org/wiki/Constructive_proof

[17] Stephen A. Cook, "*The Complexity of Theorem-Proving Procedures*", available: http://www.cs.toronto.edu/~sacook/homepage/1971.pdf

[18] A. L. Barbosa, *The Dead Cryptographers Society Problem*, unpublished, available: http://arxiv.org/ftp/arxiv/papers/1501/1501.03872.pdf

André Luiz Barbosa – Goiânia - GO, Brazil – e-Mail: webmaster@andrebarbosa.eti.br – June 2012

Site.....…... : www.andrebarbosa.eti.br
Blog.....…... : blog.andrebarbosa.eti.br

This Paper : http://www.andrebarbosa.eti.br/The_Cook-Levin_Theorem_is_False.htm
PDF……. : http://www.andrebarbosa.eti.br/The_Cook-Levin_Theorem_is_False.pdf
arXiv…… : http://arxiv.org/ftp/arxiv/papers/0907/0907.3965.pdf