Functional Hypothesis of Complexity Classes

Mirzakhmet Syzdykov

Institute of Problems in Informatics and Control,Almaty, Kazakhstan

mspmail598@gmail.com

Abstract. This work describes the hypothesis of the relation between the classes of complexity: for this purpose we define the functions over algorithms or state machines for which the equality holds true and, thus, the decision can be made towards polynomial reduction of the computational complexity of algorithms. The specific class of impractical or exponential measures of complexity against the polynomial ones is also discussed – for this case we divide these classes according to the discrete numbers which are known to the present time. We also present the approximate algorithm for the classical NP-complete problem like Traveling Salesman using the memory construction. The question of P and NP equality is important in decision-making algorithms which commonly decide inequality of these classes – we define the memory factor which is exponential and space consumption is non-deterministic. The memory consumption problem within the memorization principle or dynamic programming can be of varying nature giving us the decision to build the approximation methods like it's shown on the example of Traveling Salesman problem. We also give the notion of the past work in theory of complexity which, in our opinion, is of the same consideration in most cases when the functional part is omitted or even isn't taken into account. The model theorem with its proof of the equality of classes over congruent function is also given in the end of this article.

Keywords: computational complexity, algorithms, equality.

Introduction. The problem of deciding whether the polynomial class of finite automata can accept the solution in the same time as finding the optimal one [1]:

$$O(A_P) =? O(A_{NP}),(1)$$

where A_P is the polynomial algorithm and A_{NP} is for general case when the algorithm can be of non-polynomial type and, thus, is impractical.

The "P versus NP" problem is an old questions discussed by several communications in [2, 3, 4]. The more philosophical outcome of the problem is presented in [5].

For the past time Karp's 21 NP-complete problems were observed as a bundle with common prooperties [6]. However, in this work we define the hypothesis over complexity classes using big-O notation.

For the practical algorithms we define the class of P-complete complexity as follows:

$$O(A_P) = \{1, N, N^2, .., N^T\}, \quad (2)$$

where N is the input parameter of the *size* of the problem and T is a free parameter which is adequate to the computational environment which is presented more generally by Cook as a classical Turing tape automaton. As per our outcome, we use the definition of the algorithm as tuple:

$$A = <I, O, P> \quad , \quad (3)$$

where I is a set of input parameters, O is a set of output parameters and P stands for a procedural routine of the algorithm. Obviously the size of the problem to be computed is included in the set of inputs to the algorithm.

As our algorithm can be polynomial and, thus, complying to the computational device, the other side of NP-complexity is its classification as exponential like powerset and factorial or even Ackermann number [7]:

$$O(A_{NP}) = \{ 2^N, N! \sim N^N, \dots \} \quad (4).$$

Thus, the problem is either P- or NP-complete if:

$$O(A) = O(A_P): P\text{-}complete;\ O(A) = O(A_{NP}): NP\text{-}complete, \quad (5)$$

where A is the general algorithm which is actually the union of the accepting algorithm in P and computing in NP:

$$A = \{ A_P, A_{NP} \} \quad (6).$$

Functional Hypothesis. The pre-historical turn of our hypothesis is first met in [8] for the subset construction [9] when the functor of counter state permits the equality:

$$F(A_P) = A_{NP}, \quad (7)$$

where **F(x)** is state functor.

We state that NP-complete for problem over accepting algorithm A_P and computing algorithm A_{NP} there exist function **f** and **g** so that the following equality holds true:

$$f(A_P) = g(A_{NP}) : O(A_P) \sim O(A_{NP}), \quad (8)$$

Thus, if these functions exist we may conclude that P ≠ NP or P = NP otherwise. The decision depends on the type of the functions where as of the set (2) and (4).

For the equation (8) we can define the limits over the size of the problem if it's NP-complete, when the input parameter N is almost infinite.
For practical reasons we will consider the classical NP-complete Traveling Salesman Problem (TSP) when there exist Bellman's function [10]:

$$u_i(t) = sup \{ u_{i-1}(t - t_0) + f_i(t_0) \},\ t, t_0\ in\ R. \quad (9)$$

Our approach uses the incremental method of cities inclusion and, thus, the function (9) for TSP will be as follows:

$$F(t, n) = sup \{ F(s, n - 1) + distance(s, t) \}, \quad (10)$$

where **s** and **t** are cities and **n** is the number of the passed cities to the current step of algorithm.
The equation (10) is also known as Nearest Neighbor Heuristics [11] and is polynomial in the number of steps required for the algorithm.
However this is not principal where to find exact solution, so there is an approximate polynomial algorithm for TSP which uses ant colony optimization [12] and exact which uses exponential memory [13].
Our construction uses also exponential time and is based upon the building of network in Figure 1. Each time we visit the mediate layer we memorize the path as a context for this state and push it back to the stack of the candidates with the value equal to the path in source graph between two nodes. This allows us to adjust the maximum size of the candidate stack and, thus, get an approximate solution within the predefined memory consumption which isn't allowed in algorithm Held-Karp's algorithm [13].

We state that memory consumption is vital in memorized computations of NP-complete problem, this case is partially described in [14] where the hardware architecture design is proposed for effective computations.
The memorized computations, thus, gives us the decision to find the solution over the limited set.

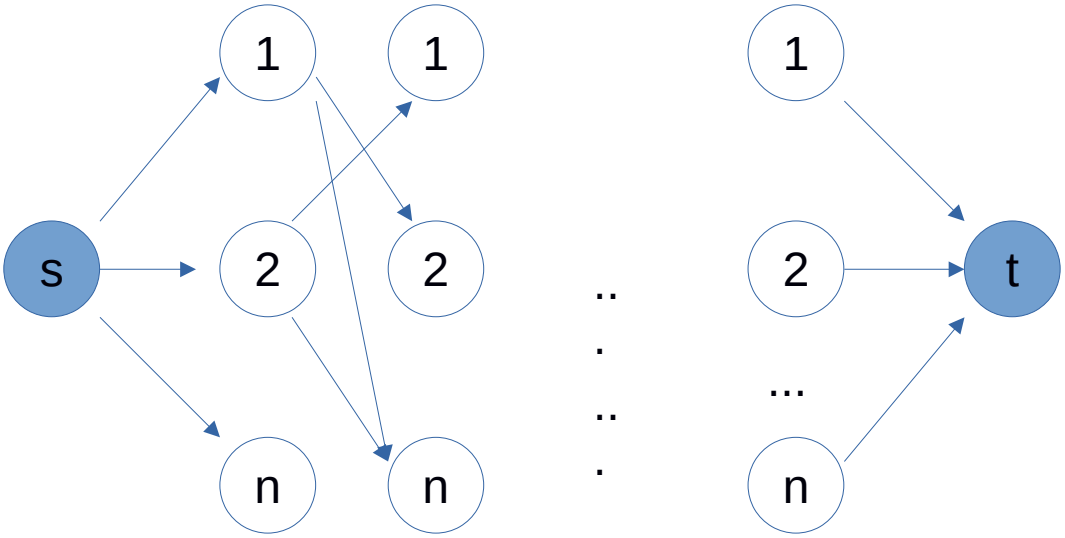

Figure 1 – TSP network for approximate solution

As to our hypothesis the following statement always holds true:

$$P \to f(x) \to N\,P, \quad (11)$$

where in (11) the P and NP are classes of complexity and f(x) is an arbitrary function according to the fact of congruence of the functor f*(x)* in counter state in the subset construction of the modified non-deterministic finite automaton, NFA, to the deterministic finite automaton, DFA, in [8].
This idea was first devised from the proof of existence of conversion between NFA and DFA made by Rabin and Scott in late 50's [9], who were also awarded the Turing Award for their proof of concept.
Thus, our main theorem based upon the fact that the NFA-counter in AND-construction algorithm is for the function f(x) in (11) and *P* and *NP* are incoming NFA and resulting DFA similarly, can be called as the theorem about congruence relations between P and NP-classes, where as the following inequality holds true, as the P-class is rather smaller than NP-class:

$$P \ll NP. \quad (12)$$

The equation (12) is practically the common case when we state that *P* is in *NP*, however, due to decision-making fact we state that:

$$f(P) = NP. \quad (13)$$

The equation (13) is according to the modified subset construction in our algorithm presented in [8].
We say that the P- and NP-classes are decidable if there exists another inverse function so that the following holds true:

$$f^{-1}(NP) = P. \quad (14)$$

The above equation devised from the equation (13) gives the prominent fact about decidability of NP-classes. Thus, we have to conclude by induction that the equation (15) holds true if we decide that our NP-complete problem is of "over" NP-class, meaning that the it's too big of almost Ackermann numbers in function [7].

$$P = NP. \quad (15)$$

The facts (14) and (15) can be proved due to the fact of the congruence of the function f(x) along the P-class and over to NP-class, seen in the main result like in equation (11).

Thus, according to our proofs of the decision against P- and NP-classes we can write our conclusion in the following form:

$$\begin{cases} P \neq NP : P \ll NP \\ P = NP : \exists\ F^{-1}(x) \end{cases}. \quad (16)$$

In parallel computing there is the following hypothesis according to the following equation:

$$\lim_{N \to \infty} \frac{NP}{N} = P, \quad (17)$$

where the N is the number of concurrent processes working on the parallel machine: the author supposes that with some type of generalization this could be even Turing machines or any other.

Model theorem. The problem of equality of P- and NP-classes of complexity is a Millennium Prize Theorem as proposed by Clay Mathematics Institute. Besides this fact, this is question when dealing with decision-making in solving problems like, for example, sorting. We will show that it's solvable by back-reference matching in regular expression. Our further material is based upon the regular expression and automata theory.
Let's define the P- and NP-classes as follows:

$$P = \{ N: O(N) > o(N) \}$$

$$NP = \{ N! : O(N!) > o(N!) \}, \quad (18)$$

for example, if we deal with factorial classes of complexity.

Further we will state the model theorem around the congruence of P- and NP-classes, these theorem is borrowed from the original source which was developed by an author in late 2015 [8]. This theorem is model, further we show by proving by induction that P != NP basing upon existence of such model classes in finite automata theory.

In [8] the author deals with the problem of matching against two expressions on (non-)deterministic finite automata for AND-operator. This is logical operator meaning the intersection of languages.

For our model theorem let's define some assumptions:

Definition 1. The non-deterministic finite automata, NFA, is a tuple like:

$$NFA = <A, S, Start(S), Finish(S), S \times (A + \{epsilon\}) \times S>, \quad (19)$$

where A is a set of alphabet, S is a set of states, Start(S) and Finish(S) is a set of starting and final states, epsilon stands for an empty sign and S x (A + {epsilon}) x S is a set of transitions.

Thus starting from state from set Start(S) we match against the word on a finite tape (probably infinite) until reaching the state from set Finish(S) by the transitional reach on the transitions connecting states by the marks, some of which could be empty.

Later we will show the congruence of non-deterministic and deterministic automata over subset construction, thus proving the properties of DFA as to be atomic in some sense. Thus, the deterministic finite automata is tuple like:

Definition 2. Deterministic finite automata (DFA)

$$DFA[1] = <A, S, Starting, Finish(S), S \times A \times S>, \quad (20)$$

where A is a set of alphabet symbols, S is a set of states, Starting is a starting state from set S, Finish(S) is a sub-set of S defining the finishing points on automaton, S x A x S is a set of transitions (please note, without epsilon like in NFA).

Thus, besides like matching in NFA, matching in DFA is linear, starting from single source point and going through all the elementary transitions for a single alphabet symbol until reaching the final state marked by the Finish(S)-set.
In elementary steps of decision the matching by DFA is rather process of function devising than, like in NFA, full simulation:

F(Starting, x[1]) = S[2];

F(S[2], x[2]) = S[3];

…

F(S[n], x[n]) = S[n + 1] (iff in Finish(S), then automata accepts the word x[1]...x[n]).

Full simulation of NFA plays a vital role in making rounds to detect if the word lays in the final accepting state, same as in case of DFA.

Now as we have defined the main clusters of the model theorem, we have to show the congruence of P- and NP-classes around our problem of matching against AND-operator as it was stated previously in [8].

For the languages L(r), where r is the regular expression, we define the new operation along with all others like intersection of languages:

L(r[1]*r[2]) = {r[1] r[2]},

L(r[1] | r[2]) = L(r[1]) + L(r[2]),

L(r*) = L(epsilon) + L(r) + L(r*r...),
L(epsilon) = {epsilon},

L(r[1] & r[2]) = L(r[1]) x L(r[2]) – the new operator by intersection of languages for regular expressions.
The building of NFA for L(r[1] & r[2]) is same in [8] as for pairing entities despite of that fact that we define the counter, which is a function in model theorem. For this purpose we define an extended NFA:

$$NFA\ Ext[1] = \langle A, S, Start(S), Finish(S), S \times A \times \{1, 2\} \times S \rangle, \qquad (21)$$

where A is a set of alphabet, S is a set of states, Start(S) and Finish(S) are starting and finishing states, S x A x {1, 2} x S is a set of transitions along the counters until the counter becomes active when simulating the matching of word on an infinite tape.

We can prove that along with subset construction [9] that our simulation leads to the construction of an atomic DFA if we would simulate these as per subset construction algorithm.

Thus the extended DFA is a subset of an extended NFA within the AND-operator along the counter function as a regulator of simulating processes within a subset construction:

$$DFA\ Ext[1] = Subset\ Construction\ (NFA\ Ext[1], Counters = \{1, 2\}). \qquad (22)$$

As we get to the NFA state during the subset construction it becomes active along the counter value which is polar to the number of incoming edges to that point.

Let us reverse to our model theorem, while we have composited main entities for better evaluation of the theorem. Basically, the theorem is based upon that fact that there exist an algorithm (in our case this is a modified by a counters subset construction) so that there would exist an in-equality of P- and NP-classes of complexity (P != NP). According to our subset construction there exist a counter-function along which the class of in-equality P standing for non-deterministic finite automaton NFA is raised to the class of NP standing for deterministic finite automaton DFA:

[NFA, P – Subset Construction(DFA, NP),

[NFA, P – Subset Construction(DFA, NP, f(x) = {1, 2}).

Thus according to the fact that there exists such f(x), so that:

Algorithm (modified subset construction):

$$f(NFA) = DFA, \quad (23)$$

where f(x) = {1, 2) is a counter-function in modified subset construction for intersection operator in regular expression.

As there exist such example this barely proves that:

$$f(P) = NP, \text{ and } P \mathrel{!}= NP. \quad (24)$$

In the next section we will show by induction that our model theorem states not only the barely fact decision of proving outcome, but rather the upper and lower-bounds for the complexities of P and NP.

Proof. Answering the Steven Cook's question stated by him in [1], we now can decide that function f(x) matter the set of entities along which the solution of the stated problem could be verified:

T = { f(x): over set of functions F }.

By the common occasion as we have proved the existence of congruent function f(x) presented by counters in subset construction algorithm for AND-operator [8], we now can afford the generalization of the model theorem by in-equality P != NP:

$$P – f(x) \rightarrow NP. \quad (25)$$

This is how it's used to be as outlined ahead. Thus, P- and NP-classes are congruent over an arbitrary function f(x), when class P is smaller (same as by definition) than class NP and, thus, limits to it over this function.

Conclusion. We have defined the set of polynomial and non-polynomial functions according to the computing complexity of the environment as they can differ from tape machines to non-deterministic finite automata and there could be the functions according to which the complexities as of big-O notation may equal and the problem, thus, can be solved using applicable or non-applicable or impractical approach.

By giving the proof to our model theorem we state the co-existence of the congruent functions for which the relation between classes can be devised.

The question is open for finding the functions satisfying the condition of equality of complexity classes.

Another question is the relation between memory and time for effective computations as the results can be dynamically computed and stored.

References

1. Cook, Stephen. "The P versus NP problem." *The millennium prize problems* (2006): 87-104.

2. Fortnow, Lance. "The status of the P versus NP problem." *Communications of the ACM* 52.9 (2009): 78-86.

3. Sipser, Michael. "The history and status of the P versus NP question." *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 1992.

4. Cook, Stephen. "The importance of the P versus NP question." *Journal of the ACM (JACM)* 50.1 (2003): 27-29.

5. Aaronson, Scott, and P. Is. "Is P versus NP formally independent?." *Bulletin of the EATCS* 81.109-136 (2003): 70.

6. Filar, Jerzy A., Michael Haythorpe, and Richard Taylor. "Linearly-growing reductions of Karp's 21 NP-complete problems." arXiv preprint arXiv:1902.10349 (2019).

7. Ackermann, Wilhelm. "Zum hilbertschen aufbau der reellen zahlen." *Mathematische Annalen* 99.1 (1928): 118-133.

8. Syzdykov, Mirzakhmet. "Algorithm to Generate DFA for AND-operator in Regular Expression." *International Journal of Computer Applications* 975 (2015): 8887.

9. Rabin, Michael O., and Dana Scott. "Finite automata and their decision problems." *IBM journal of research and development* 3.2 (1959): 114-125.

10. Bellman, Richard. "Dynamic programming." *Science* 153.3731 (1966): 34-37.

11. Kizilateş, Gözde, and Fidan Nuriyeva. "On the nearest neighbor algorithms for the traveling salesman problem." *Advances in Computational Science, Engineering and Information Technology.* Springer, Heidelberg, 2013. 111-118.

12. Dorigo, Marco, and Gianni Di Caro. "Ant colony optimization: a new meta-heuristic." *Proceedings of the 1999 congress on evolutionary computation-CEC99* (Cat. No. 99TH8406). Vol. 2. IEEE, 1999.

13. Held, Michael, and Richard M. Karp. "A dynamic programming approach to sequencing problems." *Journal of the Society for Industrial and Applied mathematics* 10.1 (1962): 196-210.

14. Traversa, Fabio Lorenzo, et al. "Memcomputing NP-complete problems in polynomial time using polynomial resources and collective states." *Science advances* 1.6 (2015): e1500031.