ABOUT THE LIMITS OF (BIO)INFORMATICS WITH SOME ILLUSTRATIONS FROM DNA AND MEMBRANE COMPUTING

Acad. Gheorghe PĂUN[1]

gpaun@us.es; curteadelaarges@gmail.com

ABSTRACT

This is an informal, mainly autobiographical discussion about a series of limits-frontiers-borderlines appearing in computer science, often addressed in natural computing, in particular in bio-computing. One only mentions such limits dealing with the competence and the performance of computing models and of existing computers, as well as other "impossibility theorems" known in the literature.

KEYWORDS: Turing computability, Turing-Church thesis, Gandy theorem, *P* versus *NP*, DNA computing, membrane computing, Conrad theorems, no free lunch theorems.

1. Introduction

The topic announced in the title above is a subject for a book, here I will only mention a few ideas and references. The semantics itself of the notions *limit*, *frontier*, *borderline* should be clarified, but I will rely on their intuitive meaning, as well as on the examples considered below.

An explicit and systematic discussion about limits is not very usual in computer science, although the limits are present everywhere and most researches, in particular the unconventional models of computation, are directly motivated by such limits met by the existing models and computers.

Here I only mention some of these limits, together with attempts to overpass them, promises, achievements and criticisms, with new limits appearing in the new frameworks – with a few illustrations from the two areas of bio-inspired computation I have worked in the last two decades, DNA and membrane computing. Three classes of limits are considered: concerning the computing power (*competence*), the computing efficiency (*performance*), and impossibility theorems (like Conrad theorems). Of course, there are many other similar limits/frontiers (just two examples: the borderline between decidable and undecidable, in automata and language theory, the borderline between universality and non-universality in membrane computing), but I will not touch them here.

The discussion is informal, but the references provided alow the reader to start exploring deeper the domain.

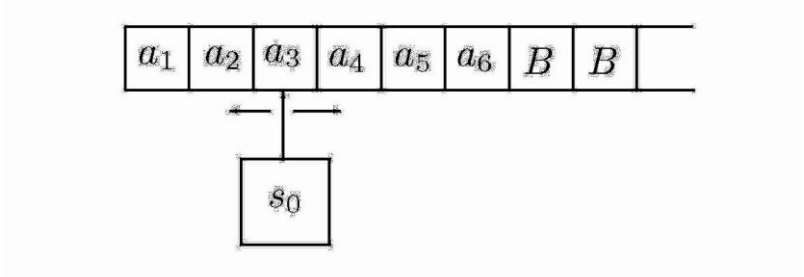
2. The Turing "Barrier"

The general framework of what follows is that of Turing computability, based on the notion of what is now called *a Turing machine,* the answer Alan Turing proposed in his 1936 PhD thesis (*Systems of Logic Based on Ordinals*, written in Princeton under the guidance of Alonzo Church) to David Hilbert question (from 1928) concerning "what is mechanically computable". Turing abstracted the way a human being computes until reaching the "minimalistic" device in the figure below: a tape infinite to the right, with a read-write head able to read a cell of the tape, under the control of a state from a finite set of states, to change the contents of the cell and the state and to move to the left or to the right. Although so simple at the first sight, this device was proved to be

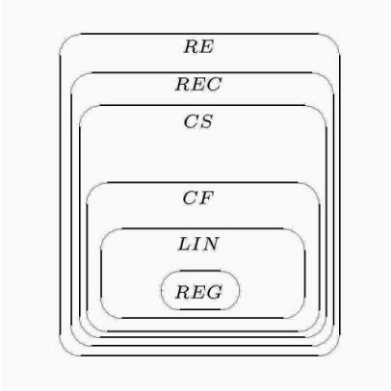
[1] Romanian Academy, Bucharest

able to simulate all computing models proposed before (by Post, Church, Kleene, Gödel) and became the standard definition of the notion of an *algorithm*.

In his thesis, Turing not only introduced his machine, but he has also provided the first example of a problem which cannot be solved by it (namely, the *halting problem:* given an arbitrary Turing machine, is there another Turing machine which can say whether the arbitrary machine halts when starting with an arbitrarily given input on its tape?) and, furthermore, proved the existence of a *universal Turing machine*, a fixed one able to simulate any particular Turing machine as soon as a code of the particular one is placed on the tape of the universal machine (this suggested to John von Neumann the architecture of the programmable computers he has constructed at the beginning of 1940). See also (Turing, 1936).



The next figure suggests a "map" of computability, in the form of *the Chomsky hierarchy*, with the class of Turing computable functions (languages, numbers, decidable problems) denoted by *RE* (from "recursively enumerable"). The smallest class, *REG*, denotes the family of regular languages, corresponding to the computing power of *finite automata*, the most restricted class of Turing machines.



These are the two "poles of computability": according to the *Turing-Church Thesis*, *everything that is algorithmically computable can be computed by a Turing machine*, that is, *RE* is the largest class of computable functions/numbers/languages.

Actually, the thesis has several versions, and there are several papers discussing it. I mention here only (Doyle, 2002).

Can we pass beyond the "Turing barrier"? This possibility was named *hypercomputing*, and there are many proposals, more than one dozen, of how we can "compute the uncomputable". Here are only three references: (Ord, 2002), (Ord, 2006), (Syropoulos, 2008).

Interesting enough, Turing itself proposed a way to compute more than his machine, by considering *Turing machines with oracles*. In the meantime, many other ideas were examined: coupled Turing machines, networks of Turing machines, oracles via (quantum) randomness, accelerated machines/processes, infinite time Turing machines, neural networks with real numbers as weights, Turing machines with an infinite input, and so on and so forth. The book (Syropoulos, 2008) provides details and references.

However, all these are considered by Martin Davis *tricks* – something uncomputable is introduced in the machine and then the machine is proved, e.g., to solve the halting problem for Turing machines, which already Turing proved that this is a unsolvable problem: (Davis, 2004), (Davis, 2006).

As expected, these criticisms raised counterarguments from the people involved in hypercomputing, see, e.g., (Sundar and Bringsjord, 2011).

Anyway, biocomputing brings new ideas and motivations in this area. One of the ideas is suggested by the strategy used in 1994 by Leonard Adleman, in his history making experiment of solving the Hamiltonian Path Problem (known to be **NP**-complete, hence intractable for the Turing machine) in linear time by using DNA molecules, (Adleman, 1994).

This was a great achievement, a *demo* that DNA can be used as a support for computing (Hartmanis, 1994), the start of DNA computing as a branch of natural computing. Actually, at the theoretical level, DNA computing started in 1987, when T. Head introduced his *splicing* operation, as a language theory model of the recombinant behaviour of DNA molecules, (Head, 1987).

In the Adleman experiment one starts by producing all possible paths in a graph (in the form of DNA molecules), then one filters out molecules which do not encode Hamiltonian paths. Otherwise stated, in terms of languages, one starts from a given set of strings and one removes strings until reaching the solution, that is, one removes the complement of the solution. This is the idea of *computing by carving*, proposed in (Păun, 1997).

In this way, we can compute languages outside *RE*, because the family *RE* is not closed under complement.

Actually, a "reasonable" way to "carve" is proposed in the mentioned paper: one starts from a regular language (hence easy to compute) and we repeatedly remove a sequence of regular languages which are linked to each other in the following way: the first language is given, the next one is obtained from the first one by means of a sequential translation (the simplest kind of transducers), and so on.

Formulated as a theorem, the conclusion is that *a language is computable in this way (by carving) if and only if it can be written as the complement of a recursively enumerable language.*

Also membrane computing can suggest hypercomputing ideas.

I only mention that membrane computing is a branch of natural computing initiated in (Păun, 1998), with a rapid development, (Păun, 2002). It abstracts computing models, usually called *P systems*, from the architecture and the functioning of the living cells. A comprehensive presentation of the domain can be found in (Păun, Rozenberg, and Salomaa, 2010), with news available at the domain webpage http://ppage.psystems.eu.

The cell membranes have two main roles in the cells: to enclose „protected reactors", with a specific biochemistry, and to facilitate the collision of reactants. This means that smaller the space, faster the reactions. In the style of accelerated machines (which perform the first step of a computation in one time unit, the second one in half of a time unit, and so on, decreasing by two the external time needed to perform each next step), we can assume that the reactions in a membrane inside another membrane are twice faster than in the upper membrane. Generalizing this assumption, we get a way to solve the halting problem: (Calude and Păun, 2003).

In both cases (carving and accelerating by creating membranes inside membranes) we pass beyond the family *RE*, the biology motivates/supports the ideas, nothing infinite or uncomputable was introduced in the model itself, but still Martin Davis is right: in both cases the process should be infinite (the carving and the hierarchy of constructed membranes). If we stop computing after a finite number of steps, then we remain inside *RE*.

I end this section by mentioning an important result concerning the Turing barrier, namely, *Gandy's Theorem*, from (Gandy, 1980).

Robin Gandy was a student and collaborator of Turing who tried to get an abstract description of a "computable device". He coined four principles (*The finiteness of description, The principle of limitation of hierarchy, The principle of unique reassembly, The principle of local causality*), formulated in algebraic terms, and proved that *whatever can be calculated by a device satisfying principles I – IV is Turing computable.*

These principles are general enough to support Martin David opinion about hypercomputability, but they also suggest ways to go beyond Turing; for instance, the last principle suggests that a machine able of transmitting instantaneously signals at an arbitrary distance (on its tape) could be able of hypercomputation.

3. Feasible versus Unfeasible (or P versus NP)

The competence (computing power) is important, in particular, the equivalence with Turing machines is important practically (this brings "for free" the universality, hence the programmability of the computing device), but in applications it is still more important to know the resources (space and, mainly, time) needed in order to compute something. This is the motivation behind the powerful theory of computational complexity developed in computer science, a framework where problems which can be solved in a time polynomial with respect to the size of the input are considered *tractable* and the problems requesting an exponential time (for the known algorithms) are considered *intractable.* The two classes of complexity are denoted by **P** and **NP**, respectively. The inclusion of **P** in **NP** is obvious, the question whether or not **P** = **NP** is probably the most important open problem of computer science. This is also confirmed by the fact that this problem is the first one in the list of seven "Millennium Problems" compiled by the Clay Mathematics Institute (www.claymath.org) in 2000, which provides a prize of one million dollars for a solution.

Such a solution can be of various forms: strict inclusion, equality proved in a nonconstructive manner, constructive equality with huge coefficients and exponents of polynomials, constructive equality with reasonable coefficients – the consequences for practical computing increases in this order.

However, the expectation is that **N** is not equal to **NP** and then, with this assumption in mind, one looks for ways to pass over the "feasibility barrier"; imitating the term *hypercomputing*, the term *fypercomputing* was proposed in (Păun, 2012) for such an achievement.

As mentioned in the previous section, DNA computing started with such a promise, illustrated by Adleman's experiment. The strategy is based on trading space for time, making use of the fact that DNA molecules are very efficient data supports, of a nanometric size. However, exponentially many DNA molecules, as in Adleman's experiment, do not really help, as soon observed, (Hartmanis, 1995).

This is the case in today DNA computing: many successful computing experiments were reported, but all of them are dealing with toy problems. Scaling-up to the level of problems of a practical size looks unfeasible (this could request huge amounts of DNA). Further details (as well as many theoretical developments, based, e.g., on the Head splicing operation) can be found in the monograph (Păun, Rozenberg, and Salomaa, 1998).

Many theoretical ways to solve **NP**-complete problems in a polynomial time, hence to obtain fypercomputations, are also provided by membrane computing, again trading space by time. In this framework, the exponential space is obtained by means of biologically inspired operations, such as membrane division, membrane creation, string replication. Another interesting idea is to start with arbitrarily large pre-computed resources, without containing "too much" information (not

to be possible to hide the solution of the problem there and then to claim that it is obtained through a simulated computation), to introduce the problem in this given pre-computed space, then to activate as much space as necessary, and solve the problem (this is the way the brain and the liver are functioning).

Like in the case of DNA computing, also in this case the Hartmanis criticism is valid, as we need an exponential working space (created during the computation, through bio-like operations).

Interesting enough, the creation of new, exponentially many membranes (or strings) cannot be avoided. This is stated by the so-called *Milano Theorem* from (Zandron, 2001).

From a practical point of view, another strategy is much more useful: looking not for exact optimal solutions, but for approximate solutions, known to be good with a specified probability. This is the strategy of a very developed area of natural computing, namely *evolutionary computing*. It contains a large number of classes of algorithms, usually based on a search in the space of solutions, making use of the brute force of the existing computers, with the search conducted in a way inspired from biology. Here are some classes of such approaches: genetic algorithms, immune computing, ant colony algorithms, bee colony algorithms, swarm computing, water flowing algorithms, cultural algorithms, cuckoo algorithms, strawberry algorithms, firefly algorithm, etc. etc.

These algorithms have a lot of practical applications, but also this approach has a drastic limitation, first provided in (Wolpert and Macready, 1997): informally formulated, the no free lunch theorem says that *all approximative algorithms are equally good* (one can also read "equally bad"), for each of them there are problems for which the provided solutions are bad.

4. Further Limits

Computer science contains also further limits. A typical one is pointed out by *Conrad Theorems* – see references in (Conrad, 1988).

The main theorem says that *a computing system cannot at the same time have high programmability, high computational efficiency, and high evolutionary adaptability.* It is rather possible that further similar "impossibility theorems" can be proved (if too many conditions are simultaneously imposed, then no computing model exists which satisfies all of them).

Such theorems are probably possible also in more general frameworks, for instance, in biological modeling, where the user (the biologist) asks for many features (understandability, scalability, adequacy to reality, relevance), at the same time with the computer scientist (easy programmability, efficiency) – and it is possible that no model exists to fulfill all requests.

Still more general: some classic experts of artificial intelligence, such as R. Brooks and J. McCarthy, have even estimated that it is possible that the current mathematics itself is not sufficiently developed for modeling such subtle notions like life and intelligence, and a new stage of mathematics is necessary for making significant progresses in this area.

About such limits of mathematics has warned us also acad. Mircea Maliţa, already in *Aurul cenuşiu* (*The Grey Gold*), vol. II, Ed. Dacia, Cluj-Napoca, 1972. And, the limits of mathematics extend also to (theoretical) computer science.

References

1. L.M. Adleman: Molecular computation of solutions to combinatorial problems. *Science*, 226 (November 1994), 1021-1024.
2. C. Calude and Gh. Păun: Bio-steps beyond Turing, CDMTCS Research Report 226, Univ. of Auckland (November 2003), and *BioSystems*, 77 (2004), 175-194.

3. M. Conrad: The price of programmability. In *The Universal Turing Machine: A Half-Century Survey*, R. Herken, ed., Kammerer and Unverzagt, Hamburg, 1988, 285-307.
4. M. Davis: The myth of hypercomputation. In C. Teuscher, ed., *Alan Turing: Life and Legacy of a Great Thinker*, Springer, 2004, 195-211.
5. M. Davis: Why there is no such discipline as hypercomputation. *Applied Mathematics and Computation*, 178, 1 (2006), 4-7.
6. J. Doyle: What is Church's Thesis? An outline. *Minds and Machines*, 12 (2002), 519-520.
7. R. Gandy: Church's Thesis and principles for mechanisms. In *The Kleene Symposium*, J. Barwise, H.J. Keisler, and K. Kunen, eds., North-Holland, 1980, 123-148.
8. J. Hartmanis: About the nature of computer science. *Bulletin of the EATCS*, 53 (June 1994), 170-190.
9. J. Hartmanis: On the weight of computation. *Bulletin of the EATCS*, 55 (February 1995), 136-138.
10. T. Head: Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49 (1987), 737-759.
11. T. Ord: *Hypercomputation: Computing More Than the Turing Machine.* Honours Thesis, Dept. of CS, Univ. of Melbourne, September 2002.
12. T. Ord: The many forms of hypercomputation. *Applied Mathematics and Computation*, 178 (2006), 143-153.
13. Gh. Păun: (DNA) Computing by carving. *Technical Report CTS-97-17*, Center for Theoretical Study at Charles Univ. and the Academy of Sciences of the Czech Republic, Prague, 1997, and *Soft Computing*, 3, 1 (1999), 30-36.
14. Gh. Păun: Computing with membranes. Turku Center for Computer Science-TUCS Report No 208, 1998 (www.tucs.fi), and *Journal of Computer and System Sciences*, 61, 1 (2000), 108-143.
15. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
16. Gh. Păun: Towards "fypercomputations" (in membrane computing). In *Languages Alive. Essays Dedicated to Jürgen Dassow on the Occasion of His 65 Birthday*, H. Bordihn, M. Kutrib, and B. Truthe, eds., LNCS 7300, Springer, Berlin, 2012, 207-221.
17. Gh. Păun, G. Rozenberg, and A. Salomaa: *DNA Computing. New Computing Paradigms.* Springer, Berlin, 1998; Springer, Tokyo 1999; Tsinghua Univ. Press, Beijing, 2004; Mir, Moskow, 2005.
18. Gh. Păun, G. Rozenberg, and A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford Univ. Press, 2010.
19. N. Sundar G. and S. Bringsjord: The myth of "The myth of hypercomputation". *Proc. of Combined P\&C2011/HyperNet 11*, May 2011, 185-196.
20. A. Syropoulos: Hypercomputation: Computing Beyond the Church-Turing Barrier. Springer, Berlin, 2008.
21. A.M. Turing: On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, Ser. 2, 42 (1936), 230--265; a correction, 43 (1936), 544-546.
22. D.H. Wolpert and W.G. Macready: No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1, 1 (1997), 67-82.
23. C. Zandron: *A Model for Molecular Computing: Membrane Systems.* PhD Thesis, University of Milano-Bicocca, Milano, Italy, 2001.