Looking behind the curtain of disruptive discourse -a personal view

Liesbeth de Mol

Looking behind the curtain of disruptive discourse – a personal view.

Liesbeth De Mol[1]

To start, I would like to make clear that I will avoid use of the term algorithm. The reason for this has to do with the shifting meanings of that notion: while originally inscribed in a history of mathematics, where its meaning steadily converged with that of method and procedure, it only became popularized in the 1960s in the context of programming where work on so-called algorithms was a way to make the discipline more scientific (read: mathematical). However, in that context, "algorithm" was soon conflated with "program". That brings me to the current meanings of "algorithm" as used in phrases like "algorithmic knowledge production"; "algorithmic bias". Here, clearly, the signification of 'algorithm' extends from the stepwise recipe-like procedure to the system of programs and data that underly parts of technological infrastructure.

While we can observe this as just another step in the history of the term "algorithm" it is one that is not innocent, exactly because the notion's history implicitly carries over certain properties that are typically associated with mathematics-(think for instance of trustworthiness or objectivity) and which the reality that is covered by the current notion clearly does not have.

Now, you could say that "of course, we know that..."; and it is just a word, but words have consequences on how we think about the realities denoted by those words. Thus, speaking for instance of "algorithmic knowledge production" *is* problematic since it hides the fact that what we are *really* talking about is not just mathematics, but also, a technology that, by consequence, involves humans and their contingencies. In fact, as I will argue here, it is only by including humans in the equation that we can identify some basic aspects of programs and their use in the production of "knowledge".

It is well-known that the first computers were embedded in a context of applied mathematics. The computer was useful since it allowed automation of computational processes which meant to compute things that humans were not capable of computing. Initially, this was largely due to the high speed of these new computers. That speed created a fundamental time gap between humans and computers which immediately led to a basic problem: how are we supposed to have control over a process that we cannot predict?

This time gap, I propose, is an instance of a more fundamental gap which is existential and perhaps trivial: that is, computing machines are *not* humans. Anchored in this is what I would call the programming problem: to find a good way to control and communicate with a device that is not human and, by consequence, does things differently from humans.

The increased need for ever more sophisticated programs resulted in a constant demand for new programming techniques. One basic approach was to make programming "easier" for humans – it goes hand-in-hand with ever more involved techniques that automate the programmer, that is automatic programming. Indeed, since the late 1940s, more and more programming "skills" have been delegated to the machine. It can be seen as the other side of the programming problem: to automate and control humans – in this case, programmers – in order to extend the usability and efficiency of computers within society.

This ideal of a world of programs that need not be programmed goes hand-in-hand with a historical development whereby people who are not programmers – clients and users – should be able to exploit computer programs. Programs should be usable by illiterate users. This is has resulted in a complex system of programs, where we have layers upon layers, from the machine level up to the Graphical User Interface, that are hard to render transparent. This only worsened by the fact that often these layers are consciously made inaccessible to users. This is a clear instance of

1 This short text was presented at the online conference *Algorithmic Knowledge Production* organized by the Collegium Helveticum, ETH Zürich, October 8-9, 2020, talks are available from: https://www.tg.ethz.ch/produkte/schaufenster/algorithmic-knowledge-production/

software's opacity – or, to put it in the words of some historians – software's historicity. From that perspective, the original programming problem has only deepened.

An interesting countermovement here, that aims to go against aspects of that opacity, is the literate programming approach that was first promoted by Donald Knuth. In his view, programs were explicitly not just about computers but also about humans who need to be able to understand programs written by others, and I quote: "*Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do*." (Knuth 1984)

While the programming problem asks for a means to control and communicate with the machine, the very fact that the machine is doing things we cannot do was also seen as an opportunity to gain new scientific knowledge. One interesting example here is the work by Derrick H Lehmer, a number theorist and computer pioneer. For him, the computer opened up new possibilities to explore a universe of discourse that was not accessible before. Thus, to him, new mathematical knowledge was only possible if we exploit the computer's basic difference with humans:

> "I would like to speak briefly of some theorem proving programs [...] in which the human is completely out-classed in what […] are fair contests. […] The novelty of the theorems is guaranteed by the fact that the proofs are humanly impractical."

Today, we could frame this "human impractability", viz. the unsurveyability of the obtained results, as an opacity problem and one that is, from Lehmer's perspective, intrinsic to the basic gap between humans and machines. However, and as I argued before, this opacity is no longer only in the results but now also in the programs themselves. What does this mean for computer-assisted science?

Two issues arise here: first of all, given the increasing complexities of the programs we are dealing with, the problem of error in those very same programs becomes intractable. Second and perhaps more urgent: it seems that many a scientist has agreed on a system where part of the results have become inaccessible to him or her by using software that is proprietary or, simply, because he/she does not bother to look behind the specifics of the functions they use. In this way, the black box has arrived in science.

One basic question then is in how far this poses a problem. Do we really want a kind of knowledge production where much of what controls the contributions of the computer, that is, its programs, are made inaccessible to us and are, possibly, controlled by instances that are not scientific per se? Perhaps, rather than jumping on the track of the easy argument stating that you do not need to understand *how* something works in order to use it in a good way, we should carefully think about what levels of opacity we can allow for ourselves (and which ones are unavoidable) in order to critically evaluate the knowledge we produce in this way. The recent debate on reproducible research gives me hope that these questions are being taken seriously. Interestingly, it is exactly in that context that literate programming is being picked up as a possible solution to some of these issues implying that also here it is realized that programs are also for humans to be understood.

As explained, for someone like Lehmer our very *inability* to "produce" the kind of results the machine can provide, is exactly what guarantees the novelty of the result. It is for the same reason that Lehmer criticised the more common ways of theorem proving of his time, that is, *fully automatic* theorem proving. For him, these were "unfair" contests and, in fact, mere simulation problems. While this can be seen as a hopelessly outdated view, there is a genuine question here: what exactly *is* the purpose of selling an idea of programs that *are* fully automatic and, supposedly, do not require human intervention? Is that possible? Or desirable?

Today, we are overwhelmed with a disruptive discourse around so-called AI and, more specifically, Deep Learning. When Google's AlphaGo success was presented in the well-established journal Nature in 2016 it was presented as follows:

"*The machine becomes an oracle; its pronouncements have to be believed.*"

This assumes a computer for which the gap with humans *cannot* be bridged or, perhaps, *need not* be bridged. But this assumption is just part of a disruptive discourse in which these Deep Learning techniques are being proposed as yet another revolution which cannot be compared with anything else that has been before; the technology is closed off from its own history. Indeed, one can quite easily show how certain claims around Deep Learning are simply instances of an old ideal, including that of full automation of the programmer. Underneath that ideal one can find a business model where the user of programs should be illiterate and dependent or, put more positively, it should be "easy" – indeed, in the current situation, it is as if literacy is not just difficult, but impossible, as the above quote from Nature shows. In that view, it is as if humans no longer matter unless as data sets and Users. But this hiding of the human within technology is something we should consider carefully; as we know very well from research on bias in data or from the Explainable AI, hiding the human here, just means to give more control to others.

My modest proposal as a historian here is to try and see through this disruptive discourse and to try and understand what is *genuinely* new about this – and which might thus give new opportunities to science and society (and I am sure there are plenty) -- and what not. From that perspective, it is perhaps interesting to connect the problems that are assumed to be intrinsic to Deep learning, to a broader historical development – think for instance of the opacity topic – which I have here framed as originating in an existential gap between humans and machines. The consequence of that is that neither human nor machine can ever be equated to the other. If we would claim to do that, we open up the way for a discourse that *hides* rather than explicates the fundamental programming problem which is not just about the machine but also the humans inside and outside of it. Or, to put it differently, the programming problem is not just a problem for programmers and the Big Tech firms, it is also your problem. That could mean, amongst others, to develop programs that are literate not just for themselves but also for humans. If we let go of that human condition, we open up the path not so much for a HAL-like machine but rather for a science that is governed by technology and its commercial interests.

Now to end this talk. David asked us to develop a problem we have – in that spirit and based on what I have been saying here, I give you more explicitly a set of problems that are playing in the background of this talk.

- what can the history of computing bring to the future of computing and how can it help us to make sure that the science we have is not just one dictated by technological ideology?
- what is the fundamental difference between deep learning programs and other programs?
- is there anything in existing approaches against opacity for programs that could be transferred to the discourse of explainable AI?
- should we aim for a general programming literacy or should we just accept that we need not understand our tools to use them in a "good" way?

Thank you.