What is an Algorithm?

Dane Worrallo

1. Introduction

Algorithms are familiar to most people: they can play a part in almost all of the tasks in daily life, to varying degrees. However, to attempt locking down a definition of an algorithm can prove problematic for multiple reasons. Firstly, if algorithms can take any number of forms and accomplish any number of tasks, then the definition that remains to unite them will be one that is extremely general or vague, and thus of little use. Secondly, if we are to look to the algorithm's origins in mathematics and computer science for a definition, we quickly become overwhelmed with unfamiliar notation that bears little resemblance to how we interact with the algorithmic process, and would be akin to learning a new language in order to make this notation become clear.

This short paper will attempt to navigate between these two issues to clarify the general properties of algorithms and their uses in digital processes. I will avoid venturing too far into the domains of mathematics and computer science, but when I do so, I shall attempt to provide a clear explanation fit for a general audience. The paper will be divided into three main sections. In section one I will attempt to explain what an algorithm is and through an example, explore its applicability and usefulness in undertaking tasks. Section two will give a brief history of the algorithm and how the objective of various mathematicians and philosophers to provide a language or system which is able to algorithmically complete any given task, and the various limits and issues raised in doing so. The third section will look at how algorithms are used in contemporary practices, and the conceptual leap that is required between understanding the mathematics of the algorithm, to how they operate and are interacted with in practice.

2.1 The Foundations of the Algorithm

An algorithm is – to use the most general of definitions – ***a step-by-step process used to accomplish a particular task***. For example, if I wished to accomplish the task of reading a book. There are a number of steps I would have to take in order to accomplish this task: I would have to pick up the book, then open the cover, then read the first page, then turn the page, then read the second page, and so on until the end of the book, when I can close it. Figure 1 provides a clearer picture of this task.

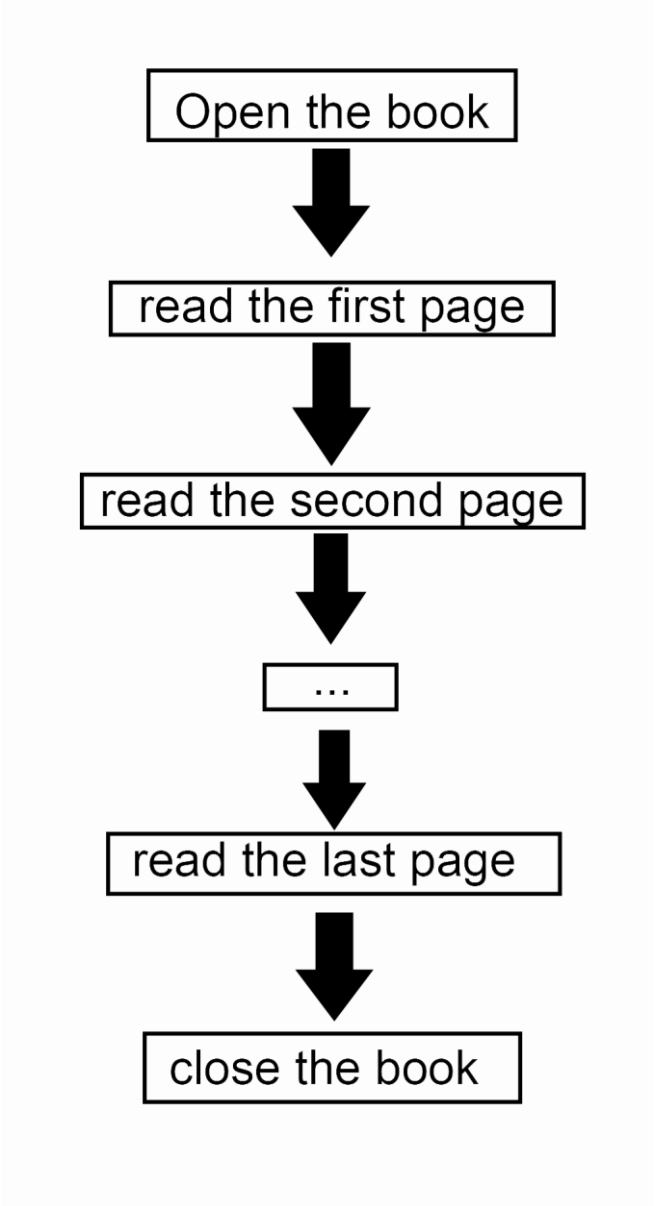

Figure 1. The 'algorithm' for reading a book.

Let us compare this figure with the definition given above: the ***task to be accomplished*** in this case is the reading of a book. The process of doing so is achieved ***step-by-step***: we begin by reading the first page, then, the second, and so on ("…" is the symbol often used that denotes this phrase) until we reach the final page, at which point the task of reading the book is accomplished. This is a very basic algorithm, but one which has succeeded in moving from state *A* (wanting to read the book) to state *B* (finished reading the book). It is a set of ***instructions*** that can easily be followed by anyone wanting to accomplish this task.

2.2 Multiple Uses of an Algorithm

To re-iterate: In its most general sense, an algorithm is a set of instructions that are able to accomplish a task. Anyone that can follow the algorithm's instructions can accomplish the task. If we were to hand someone the book used in the previous example, alongside the algorithmic instructions shown in figure 1, providing they could read the instructions and understand them, they should be able to complete the task easily. This is a key aspect of digital algorithms also: if I create a file on one computer that can be opened or read, then if I give that file to another computer to open up, it should be able to follow the same instructions and read the file the exact same way.[1] We do not need to rebuild the algorithm each time it is given to another use or computer to undertake.

We now have a conception of the algorithm that we can start building upon. The algorithms described thus far are able to accomplish one single task, as long as the user or computer can read the instructions. Algorithms are, however, of very limited use if they can only accomplish a single task and then become redundant when it is achieved. The algorithm for reading our book serves its purpose well, but what if we wanted an algorithm that could be used to read multiple books? How would we prepare an algorithm in advance that can read books it has not seen? This is achieved by making sure our ***objects*** (or books) are ***well-defined***; meaning that while each object may differ in some respect, these differences do not alter the object itself (the book does not become a car, for example). Different books will have a different amount of pages, but all of them will have pages. The number of pages will be a ***variable*** which alters how the algorithm will carry out its task. This means that any book, no matter how many pages it has, will work with the algorithm. The next step is to make sure that the algorithm is well-defined: that its instructions cannot be misinterpreted no matter the values of these variables, and that the algorithm can complete the task. Figure 2 shows how such an algorithm may be constructed.

[1] The two computers must obviously have the necessary software or be operating in the same language at some level, but this point will be addressed in more detail later.

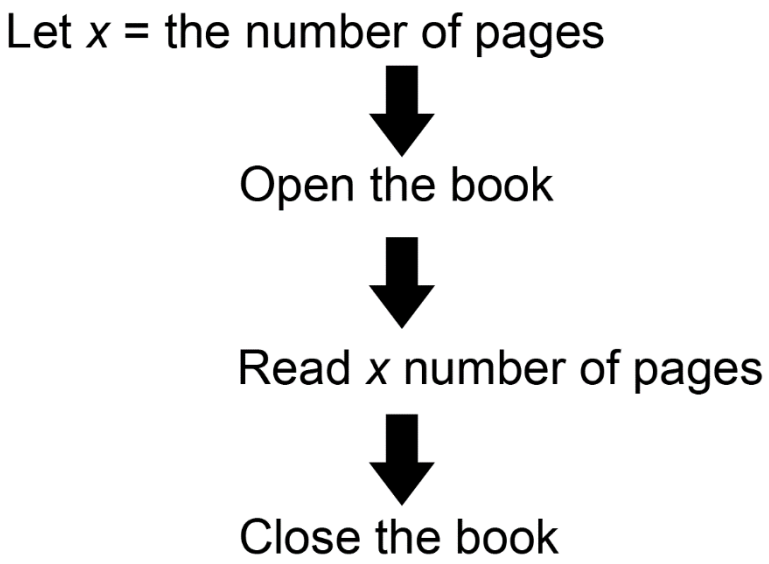

Figure 2. Algorithm for reading a book with any number of pages.

For this example, a variable ***x*** is being used to denote the number of pages in a book. In the above paragraph, we mentioned how every book may have a different number of pages, but all of them will have pages, so no matter what the number of pages in a book, it will still be a book. For example, if I have a book that is twenty pages, I let ***x = 20***, and then replacing *x* in the instructions given, I would read twenty pages before I close the book. Now I do not have to give somebody a book along with the algorithm: I could take them to a library and ask them to pick out any book they wished, and as long as they follow the instructions by substituting *x* for the number of pages in the book, they choose, the algorithm will accomplish its task.

These two examples show the potential in constructing algorithms to solve problems and complete tasks. In doing so, however, we have overlooked the difficulties that are involved in constructing an algorithm whose instructions can be unambiguously interpreted and followed without error. In giving instructions for reading books, one can run into problems such as:

- What if the person undertaking the task cannot read or know what is means 'to read'?
- What if they cannot read the language the book is written in?
- What do they do if there are images? How do they 'read' an image?

Of course, our algorithm can overcome these problems by adding in additional instructions to solve them. For example, if a book is chosen that is written in a language the reader cannot read, then they should choose another book.[2] There is theoretically no limit to the enhancements we could make to this algorithm to incorporate a solution to every foreseeable problem, but if our reader had

[2] These types of arguments that are composed of "if…then…" are ***conditional*** arguments.

to undertake an algorithm that had hundreds or thousands of steps just to check if it was a book they could read, then the task would take such a long time that it would not be worthwhile doing, or may never complete at all.[3] These problems relate to how the instructions themselves are read, and that the words, instructions and commands that comprise the languages we use every day can have multiple meanings depending on context, or may change over time. Ideally, our algorithms would be written in a language that could not be misinterpreted, and the functions of the words it used never changed: if a reader were to learn this language, they could perform the algorithms written in that language without any chance of error, and the language would be fixed regardless of context. At this point, we shall now turn to the work of the philosophers who undertook this task, and how their attempts at constructing such a language met with varying levels of success.

3.1 The Search for a Universal Language

The traditional task of Western philosophers throughout history generally involves devising a logic or system through which existence and the true nature of things can be explained. Such philosophy is the most general of enquiries, in that it pertains to all things, but also the most in depth, by penetrating to their core truth. Of particular interest are the philosopher's that attempted to build a ***universal language***, through which all truths could be expressed and calculated without error.[4] On this subject, Descartes wrote a letter in 1629 expressing interest in the idea of a universal language, and how one may be developed. He suggests that such a language would be entirely possible:

> Order is what is needed: all the thoughts which can come into a human mind must be arranged in an order like the natural order of the numbers. In a single day one can learn to name every one of the infinite series of numbers, and thus to write many different words in an unknown language. The same could be done for all the other words necessary to express all the other things which fall into the purview of the human mind.[5]

Descartes' here is using the example of numbers to support the idea that a universal language exists: no matter how numbers are said in many different languages, they all refer to the same numbers, and many use the same symbols (1, 2, 3 etc.). If numbers are universal in this way, then Descartes

---

[3] Imagine, for example, if instruction *A* told the reader to proceed to instruction *B*, which in turn told the reader to proceed to instruction *A*, which lead to *B*, and so on. The reader would be caught in a loop that render them unable to proceed with the task.

[4] The term 'universal' here means that a proposition (or language) is true regardless or time, space or circumstance: no matter where (or when) in the universe such a statement is made, it would be universal if that statement remained true and was not affected in any way.

[5] René Descartes, "To Marsenne, 20 November 1629" in *The Philosophical Writings of Descartes Volume III: The Correspondence*, trans. John Cottingham et al. (Cambridge: Cambridge University Press, 1991), 12

surmises that words would be too. He recognised that a true universal language must be able to express clearly the simplest ideas of the human mind, but this is something he did not feel he wanted to do.[6] Descartes furthermore remarks that while it is *possible* that such a universal language could exist in the way here described, he would not want it to see it ever used. Saying that "For that, the order of nature would have to change so much that the world would be turned into a terrestrial paradise; and that is too much to suggest outside of fairyland."[7] Descartes would seem to recognise that the languages that are used by people and their ambiguities are too much ingrained in the world, and that such a seismic shift is unrealistic.[8]

Descartes also states that "The greatest advantage of such a language would be the assistance it would give to men's judgment, representing matters so clearly that it would be almost impossible to go wrong."[9] As a consequence, the ambiguity of language would all but disappear, and algorithms that employed this language would not run afoul of misinterpretation as long as it is used correctly. Following Descartes, the philosopher and mathematician Leibniz pursued this idea with much more fervour, and providing what I would argue is the first example of a computational philosophy. In a 1685 letter, Leibniz discusses the method he has discovered by which "…we can represent all truths and consequences by Numbers."[10] Such a language ***encodes*** these "truths and consequences" into numbers or symbols, which in turn carry out these operations on their own, like a machine or ***computer***, which is what Leibniz believes will be the outcome of such a process:

> The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate, without further ado, in order to see who is right.
>
> If words were constructed according to a device that I see possible, but which those who have built universal languages have not discovered, we could arrive at the desired result by means of words themselves, a feat which would be of incredible utility to human life.[11]

---

[6] Ibid., 13.
[7] Ibid.
[8] As will be shown below, fellow philosopher Leibniz would have no quarrel pursuing this course with such optimism.
[9] Ibid.
[10] G.W. Leibniz, "The Art of Discovery" in *Leibniz: Selections*, ed. Philip P.Wiener (New York: Charles Scribner's Sons, 1951), 50. Leibniz is here referring to his *Dissertation of the Art of Combinations*, one of his first works in which he attempts to develop a universal language based on numbers, developing proofs including one for the existence of God. See G.W. Leibniz, "Dissertation on the Arts of Combinations" in *Philosophical Papers and Letters*, 2nd ed., ed. and trans. Leroy E. Loemker (Dordrecht/Boston/London: Kluwer Academic Publishers, 1989), 73-84
[11] Leibniz, "The Art of Discovery", 51-52

What sort of "device" is Leibniz imagining? Picture this scene: a dispute has arisen between two parties, and they cannot agree on which one of them is right. They are at an impasse, and so turn to Leibniz's "device" to provide the answer. They enter all of the variables and symbols, and the device undertakes the operations encoded within the symbols ***step-by-step*** to resolve the dispute. A universal language, for Leibniz, would be able to resolve any dispute as long as all the variables are input into the device, and would also provide an answer free from the vested interests of either party. This device calculates its response ***algorithmically***, following the instructions of the universal language, and the symbols that are a part of it, anyone can input their queries or variables and receive an answer. It is not difficult to see how Leibniz's conception of such a device is comparable to the modern ***digital computer*** in terms of how it may be used to solve all sorts of problems depending on its programming. Leibniz's belief that every problem could be solved algorithmically via such a universal language (or universal machine) required laying down foundations that – in the spirit of universality – would be applicable and avoid contradiction no matter when or where they were used.[12] How does one know that there are no contradictions in its rules, or that something unexpected may emerge in the future that cannot be encoded or computed in this language? In terms of the example used in the previous section: one can construct an algorithm that accounts for every possible contingency, or every type of book in every language so that the instructions can be followed, but will someone in the future be able to make a book that cannot be read using these comprehensive set of instructions?

3.2 Completeness (Incompleteness)

In order for a universal language to be qualified as such it must meet two qualifications: it must be ***complete***, by which is meant that there are not and will not be any new statements, or propositions that cannot be expressed or constructed within. It must also be ***consistent***, in the sense that no two statements or propositions ***contradict*** each other.[13] Producing a language or formal system that meets these two qualifications, and can *prove* it, is a task that has pre-occupied a number of philosophers and mathematicians since Leibniz. The task of this paper is not to chronicle this history,

[12] Leibniz outlined a basis for such a universal language by constructing a system through which every number would be represented through a combination of zeroes and ones: the binary or base-two number systems used in digital computers. See G.W. Leibniz, *Explanation of Binary Arithmetic*, trans. Lloyd Strickland, accessed 4 April 2019 http://www.leibniz-translations.com/binary.htm

[13] For example, a language cannot be able to prove that both $x = 1$, and $x = 2$, as this would be a contradiction, and both cannot be true.

but to position the landmarks of this development and how they relate to the effectiveness, applicability and limitations of the algorithm.[14]

In order to construct a complete and non-contradictory system, it must be able to handle any content or function that wishes to be fed into it, as well provide a unique result or output in response for it. The difficulty with numbers is that they can *mean* a lot of different things: the number "2" could relate to "2 chairs", "2 apples" etc. and the number "2" could be broken down and used in a number of ways. The solution to this is to remove meaning from the system, leaving it a functional machine in the way Leibniz envisioned, and the symbols of the machine's language are parts of this machine that perform their function without them needing to mean anything.[15] In order to overcome this ambiguity in numbers, Georg Cantor developed a system that replaced the use of numbers with ***sets***. A set essentially replaces the use of numbers in a purely logic-based system. For example, if I wanted to perform an operation using all the odd numbers, I would run into a problem insofar as there are an infinite number of them, and it would be impossible for the system to go through an infinite quantity, and would thus never complete its operation. Likewise, the issue of our 'book' algorithm cannot guarantee to function with every book unless we program each book that has or will existed; another impossible task. Instead we can perform our operations on sets: the set of all odd numbers contains an infinite sequence, but it is a single object defined in a universal language or system that can be manipulated and operated on without having to deal with its contained infinity.

This explanation of sets barely scratches the surface of this significant mathematical shift, but delving any deeper will quickly overwhelm us in mathematics, which at the outset I declared would be avoided. Let us consider this significance in terms of its impact on the effectiveness of constructing algorithms. As has been shown, the 'book' algorithm works perfectly well if we have a book at hand and we can adjust the variables (number of pages) to accommodate said book. We did however, find limitations regarding how effective the algorithm would be with regards to different types of books and the potentially infinite variables that one could encounter in picking a book at random from a library. If we replace the book in our algorithm with ***the set of all books***, then

---

[14] This history, as well as its relation to the development of the digital computer are most succinctly and accessibly detailed in the individual chapters of Martin Davis, *The Universal Computer: The Road from Leibniz to Turing* (New York/London: W.W. Norton & Company, 2000)

[15] This can be a little difficult to comprehend. Consider the act of switching on a light through the use of a button: The act of pressing a button does not mean anything, the electricity passing through the circuit does not mean anything, and the light switching on does not mean anything. Each component of this 'system' could be considered a ***symbol*** that when activated performs its function, but it doesn't mean anything when it does. Meaning *can* be attributed to these symbols or components, but it would not alter or interfere with the operations of the system itself.

anything that is classified as a book will belong in that set, and the algorithm will be able to operate with any of the objects contained therein. Returning to a point made in the first section, the set must be ***well-defined***, so in this system there must be a certain criteria that an object must meet to be classified as a book. We are reaching the limits for the usefulness of this example as we head deeper into the concepts of mathematics which do not translate easily into practical examples, but the concepts of sets are an example of how an algorithm can be used in circumstances where variations can be infinite in some capacity, such as the set of odd numbers described above, but still accomplish a function or task that accounts for these variations.

The culmination of this task to create a complete, consistent system came about through the work of Bertrand Russell and Albert North Whitehead, whose three-volume *Principia Mathematica* claimed to have fulfilled the task of completeness and consistency through which all mathematical proofs could be expressed.[16] It is a monumental system that carefully avoids the paradoxes that have plagued mathematicians before them, most notably Russell's paradox, which is:

> Does the set of all sets that does not contain itself contain itself?

And is often characterised in the following way:

> The barber of Seville shaves everyone in the town that does not shave themselves. Does the barber shave himself?

Let us explain using the latter example. If the barber *does not* shave himself, that means he – as the barber who shaves everyone who doesn't shave themselves – must then shave himself, but that also means that if he shaves himself, then he – as the barber – does not have to shave himself…and so it goes around in this fashion forever.[17] If this were an algorithm, then it would never complete its task (answering the question) because it would get caught in an infinite loop following its own instructions. The solution to this in *Principia Mathematica* is to define ***types*** of sets, and limiting the interactions between different types: essentially, the barber cannot also be a resident of Seville.

The biggest blow to *Pirincipia Mathematica* came from the mind of Kurt Gödel, whose ***incompleteness theorem*** showed that not only did a paradox exist in this system, but that *any* logical system that attempted to be both a consistent and complete account of mathematics would have

---

[16] Albert North Whitehead, Bertrand Russell, *Principia Mathematica: Volume I*, 2nd edition (Cambridge: University Press, 1968)

[17] Explained using the former statement, if the "set of all sets that do not contain themselves" (the barber) does contain itself, then it cannot contain itself, which would then mean that it would have to contain itself etc.

the same paradox within it.[18] Alan Turing similarly showed the limits of such systems in his ***halting problem***, which proved that it was impossible to construct a system which could determine whether all statements would be computable (would reach the end of their calculation) independent or in advance of their being computed.[19]

The search for a universal language or system that can express all statements of mathematics without error or inconsistency thus runs into a seemingly unsurmountable problem, and with it, the algorithms built with them will possess inherent limits with regards to what they can or cannot accomplish. With this in mind, Turing focused not on trying to surmount this problem, but having recognised the limits independent of these systems being put into practice, he focused on how these algorithms would have to work within the various practices or tasks in which it was deployed. Returning to the conundrum outlined in the introduction, we are faced with an almost limitless array of these practices, each with their own rules and limits and quirks which an algorithm must account for in its instructions or programming to 'fit in' and contribute those practices or tasks. Although the concept of a universal language has its limits as shown in this section, it is not without merit: such languages are capable of working with a large number of processes and information, as evidenced by the versatility of the digital computer in accomplishing countless tasks depending on its programming. With this in mind, this paper, in searching for a definition of an algorithm, will have to combine the effectiveness of a universal language combined with the ability to engage with specific practices.

4.1 The Ongoing Task of the Algorithm

What makes Alan Turing so influential is not just because his work in symbolic logic and mathematics made the modern digital computer possible, but that he considered how his universal machine could be programmed to accomplish many different tasks: just one computer that can be programmed in an infinite number of ways. It is a testament to his forward thinking that he envisioned how

---

[18] See Kurt Gödel, *On Formally Undecidable Propositions of Principia Mathematica and Related Systems*, trans. Brian Meltzer (New York: Dover Publications, 1992). Gödel's proofs (of which there are three) are complex, and require an understanding of *Principia Mathematica*'s operation and notation in order to understand, which is beyond the scope of this paper. Essentially, it involves constructing a statement using the systems rules and symbols that says something akin to "this statement cannot be proved in *Principia Mathematica*".
[19] Alan Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem" in *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life: Plus The Secrets of Enigma*, ed. B. Jack Copeland (Oxford: Oxford University Press, 2004), 58-90

computers would find use in an array of practices and processes, and his work is still relevant even today.

We have in previous sections focused on the algorithm as a process of completing any given task. Once the algorithm completes this task, the algorithm halts. The computer as a calculation machine also works in this way: when it finishes calculating, it can stop. But some tasks do not simply have an end: they are ongoing, with no determined end-point or destination. For example, if one has an algorithm used for monitoring the weather, there is no determinate point at which the weather ends. If, however, I want to know the state of the weather now, or for a day, a week etc. I can program an algorithm that can accomplish this by performing a check on the weather at the current time, and at regular intervals depending on the length of time and frequency I wanted to check. The term ***real-time*** is often used to describe processes that are taking place in the present moment, so if I wanted to see the weather change in real-time, I would program an algorithm that would constantly check weather data and would notify me of any changes. Next, consider how I would attain an appropriate definition of what constitutes "weather": I could monitor satellite imagery of cloud movements, wind speeds, humidity and so on. All of these are separate tasks or algorithms that may combined to give a more comprehensive and accurate account of the weather.

The starting definition for an algorithm set out in the beginning of this paper consisted of a task to be accomplished through a set of finite steps. The example of measuring or calculating the weather above complicates this definition through the means by which it works in practice. The real-time task of checking the status of the weather is – in one sense – a task that cannot be accomplished, because the weather is always changing. Next, the various processes involved in checking the weather do not proceed step-by-step: they work simultaneously or in ***parallel***, so one machine may be monitoring the wind speed while another will be measuring the humidity, and this parallel processing cannot be adequately defined as a step-by-step process because neither one necessarily needs to come before the other, and if they are not carried out at the same time, they will combine to give an inaccurate output (in this case a weather report).[20] Algorithms are used in many such tasks in order to monitor conditions, changes, and to keep people up to date with them in contemporary life, where many objects and artefacts have a digital dimension that can connect them to other

[20] In the case of weather checking, a small delay would not prove damaging to the accuracy of the output, as the weather will not shift significantly in the small gap between finding one measurement, then finding another, but there are examples where such a delay would prove costly, such as in algorithms used in the stock-markets and trading. In addition to this, there is no universal solution to deciding which measurement would be taken first. In some cases, a measurement that changes slowly would be calculated first as the intervening time between that calculation and finishing the task would mean there would be little change in a slow changing variable, but if a quickly changing variable requires an excessive amount of computation, it may be preferable to calculate it first.

digital devices. In many of these cases, we do not need to *see* that the algorithm is working, and in fact never do, as the almost instantaneous completion of task gives no time for consideration into how it is being achieved, or how defining this algorithm would benefit those that use them. If we are to answer the question "what is an algorithm?" we must continue by *making* time to consider the effects it has, and how it accomplishes its task.

4.2 Who Should Care About the Algorithm? (or, A Call for Transparency?)

Turing's conception of a universal machine (computer) was a device that, when delivered any finite input, it would be able to provide an output; as the machine would have all the necessary programming within it to complete the calculations. Furthermore, Turing's profound conception was that this machine could be programmed to have multiple computers within itself, which would allow the completion of tasks such as the weather checking example, which requires the computer to accomplish several tasks in real-time.[21] As long as there is an input and an output, the algorithm, can situate itself in between them in order to facilitate the transition from one state to the other, as shown in figure 3.

Figure 3. Diagram showing the basic process of computation. An input exists in one state, and is transformed by the algorithm to the output.

The algorithm exists in this process in order to transform input to output. In many uses and practices, the algorithm does not need to be understood by its user, only for it to carry out its task

[21] But as mentioned in the previous footnote, the complexities of the specific practices it is engaging with will require an algorithm to be programmed to provide the most accurate and effective output.

successfully: if I want to know the current weather, I do not need to see all the calculations the algorithm is made, or how it is done. The term ***black-box*** is often used to describe such a system, in which the algorithmic process is completely interiorised and out of sight: it does not contribute to or affect the task in any way, only providing the machine equipment (in the form of code and programming) to carry it out. Framed in this way, the algorithm becomes meaningful exclusively to the computer scientists and software engineers, who alone possess the specialist knowledge required to open up the black box and to tinker with the algorithm's components. To ask such a person "what is an algorithm?" they may answer using the language of the computer scientist, the logician, or the mathematician; the languages of the specialist, and their own task of making algorithms as effective and efficient as possible. Their definition, however, will not be meaningful to those without this knowledge, but still use and interact with algorithms on a daily basis just fine without this knowledge. This is not to disparage the work of the computer scientist, and certainly their work is vital to the success of algorithms incorporating themselves into the various facets of everyday life, but this search for meaning in foundations – as shown above in the search for the universal language – is at best an ***incomplete*** picture, and at least part of its meaning must be found within the practices within which it is embedded.

As mentioned above, in using digital processes to accomplish tasks, such as checking the weather, one does not need to know the inner workings of the algorithm, the logical syntax, or the programming language in which it was written: as long as it accomplishes its task successfully and without minimal disruption, one need not give the algorithm much thought. Indeed, even if one wanted to, the algorithm's process is designed to be as efficient, fast, and ***transparent*** as possible, thus obscuring *how* it is accomplishing this task, and the cost or consequences of its output. From an ethical standpoint, these costs may involve the user giving their personal details, and thus intruding on their privacy, with their details being used for purposes for which they did not give their consent. Users can also find themselves part of larger algorithms and contributing to 'outputs' they would not agree to if they were aware of their operation.

The ***transparency*** of the algorithm must be twofold: in order to be a useful and efficient participant in practices, it must fulfil its task without interfering or distracting from said task. Alongside this, there must also be transparency in how the algorithm is operating and accomplishing its task: what data it is using to complete the task, where the data came from and how it was acquired etc. These two aspects of transparency seemingly run contrary to one another, and when one is rendered more transparent, it interrupts the other. However, it is not as simple as lifting the black box to reveal the workings underneath, as Seaver states:

> At their most simple, calls for transparency assume that somebody already knows what we want to know, and they just need to share their knowledge. If we are concerned about Google's ranking algorithm for its search results, presumably that knowledge exists inside of Google […]. While transparency may provide a useful starting point for interventions, it does not solve the problem for knowing algorithms, because not everything we want to know is already known by someone on the inside.[22]

If the process of transparency is a process of seeking the 'source' from when the algorithm sprung, then, Seaver highlights, one may actually find no source: an algorithm can be made through the combination of multiple people, departments or companies, none of which can offer a definitive source or foundation, or even have had any communication with one another. Furthermore, an algorithm does not simply mirror the wishes of its creator(s): when put into practice, analysing data its creators have never seen, those practices belong to the algorithm itself, and take on a life of their own. Again we can return to Turing and Gödel's findings above, which highlighted the inherent limits of systems independent of the practices within which they are deployed: it is the various dimensions of practice that the algorithm recognises and adapts to that allows it to both distinguish itself from its creators, while also embedding itself in the practices it is engaged in.[23] Transparency such as that which Seaver mentions ignores the agency and actions of the algorithm as if it is not there, instead going after those which created it as having all the answers.

The algorithm must be acknowledged within the practices it is embedded in. Algorithms that are constantly monitoring and altering various processes in the world, such as checking the weather, have an effect on people's responses or activities, and if these processes entail an ongoing, never-ending engagement, then one cannot simply wait until the weather (for example) finishes to assess the algorithm's effectiveness and its effect on those who engage with it. This interaction and critique must take place *alongside* the running of the algorithm as an active participant in a practice, with its decisions and biases able to be accessed and assessed just as any other participant.[24] If an algorithm is doing harm to other participants, or is incorrectly producing results, the means should be available to change this, or provide new data for the algorithm to learn from. Algorithms have a direct impact

---

[22] Nick Seaver, "Knowing Algorithms" (2014): https://static1.squarespace.com/static/55eb004ee4b0518639d59d9b/t/55ece1bfe4b030b2e8302e1e/1441587647177/seaverMiT8.pdf

[23] This issue of adaptability is more significant for complex algorithms that are self-correcting, or modify their own behaviour based on what results they produce. But even for more simple algorithms, there may be a number of different algorithms that can accomplish a task, and how effectively or quickly they do so can alter the outcome to various degrees, and so even the most simple of algorithms may vary depending on these various dimensions of practice (computer hardware, amount of data, who interacts with it etc.).

[24] It is important, however, to resist the temptation to ***humanise*** the algorithm, and to judge it by its ability to grasp a task in the way a human would.

on those that use or interact with them, and it is important to remember that its outputs may present a biased or incomplete picture of the task it has undertaken.

For those that use an algorithms to accomplish a task, having the means to access how the algorithm has arrived at its output allows them to grasp at the algorithm itself, as an entity or participant in itself, and the possibilities and prejudices it can bring to a task or practice. With this conception of the algorithm as something that does not need to be transparent, but is able to be navigated into and around when it is necessary, one can navigate all sorts of algorithms by identifying the various features and processes that are common across them. To conceptualise this, we shall return once more to the work of Turing for one of his most famous examples.

4.3 The Interrogation Game

The interrogation game (also known as the imitation game) is a theoretical scenario used by Turing to show how a computer should be afforded the same status as a non-computer if they can both perform the same task without someone being able to tell them apart based on the output of that task.[25] In Turing's example, he imagines a game in which a human and computer, *A* and *B*, are given a question to respond to, and for a judge, *C*, to determine which participant is the computer and which is the human based on the responses they give. If *C* is unable to tell which responses belong to which participant, then both must be considered as ***thinking***. This is because when a human gives a response to a question, we presuppose that they have arrived at that answer through ***thinking***. If a computer arrives at a response, we presuppose that it has done so by ***calculating*** a response. If a judge cannot decide which response was produced by the *thinking* participant, then both responses must be classed as coming from *thinking* entities until a further response distinguishes the two.[26]

Turing's game serves as a warning against pre-judging a participant based solely on their ***identity*** as human or computer etc., and instead they should be judged on their responses to situations, and their contributions made through those responses. Such a position warns against discriminating

[25] Alan Turing, "Computing Machinery and Intelligence" in *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life: Plus The Secrets of Enigma*, ed. B. Jack Copeland (Oxford: Oxford University Press, 2004), 433-464. This paper will use 'interrogation game' in keeping with Turing's original term over 'imitation game', which also highlights the role of questioning and engagement present in the term 'interrogation'.

[26] Of course this argument could reverse: if the judge cannot distinguish between *A* and *B*, then then can both be considered *calculating* rather than thinking. However, the purpose of this game was for Turing to support his view that computers could be more than just calculating machines, and could have a role in all sorts of tasks given that how they could accomplish them would be indistinguishable from a human's completion of the task.

against an algorithm *just because* it is an algorithm, but it also means that one judges the algorithm on its outputs rather than the process by which it arrived at its conclusion (i.e. the contents and operation of the algorithm itself). In this context, just as one may deduce what someone is thinking based on a response, so one will be able to work out how an algorithm is producing its responses. This sameness in status of both participants – as thinking participants – is not just an accepted status quo: the aim of the interrogation game is to ask questions that will betray which responses belong to which participant. The status of thinking is never just accepted (like the identities of human and computer), but are under constant review. In our everyday dealings with algorithms, we will become aware of the algorithm's presence or its operation as a distinct object or process if it does not accomplish its task or provides an unexpected result. If one checks the weather via an application on their smart phone, and its response is incorrect, one is left to wonder to how it arrived at that result. It may be a software error, incorrect data, sudden changes which have not yet been detected by the algorithm and so on. Likewise, if we ask a person what the weather is like outside, and they are incorrect, we may consider reasons why they got it wrong (of course, we could also just ask them).

Time and again, our critique of the algorithmic process seems to also apply to humans performing the same task, which further reinforces Turing's aim to offer the same status to each (in this case, *thinking*) when playing the same game. However, we should afford specific attention to the algorithm in terms of how it is programmed: a human is not constructed to think or respond in a certain way, or more specifically, in a language or system which we can so easily decipher and understand (it must have been developed by somebody – singular or multiple). Accessibility to the algorithm's operations and logic allows a user or one who interacts with it to see just how its task is accomplished. If we are constantly aware that we are dealing with algorithms in all aspects of contemporary life, does that mean that Turing's interrogation game is redundant? Not so. Even though we are aware we may be asking questions to a computer, such as through a web search engine, we expect an intelligent and coherent response, and because we recognise it as an algorithm, we should be aware of its limitations. The interrogation game does not end when we can distinguish the human and the digital, but rather necessitates its continuation: as new data and practices emerge, we must interrogate the algorithm to ensure that it does not introduce a 'digital divide' between an algorithm's response and that of a human. This is important to avoid installing an authenticity to one side or another: either the algorithm's response can be dismissed as a ***simulation*** or ***imitation*** of a 'real' version of a process, but also we must avoid conferring on the algorithm an 'objectivity' or an ***unquestionable*** truth that is somehow purged of the subjectivity or bias of human

thought.[27] In attributing a sameness (thinking, or otherwise) to each of the participants, their relationship to one another can continue to be reconfigured, their needs assessed and met, and their challenges overcome.

5. Conclusion

In order to answer the question "what is an algorithm?" and to attribute some form of meaning to it, there must be certain groups, people or parties to have an interest in the algorithm's various aspects: whether that be its programming, its cultural impact, or simply those who want to check on the weather via their smart phones. The algorithm means something different to each of these groups, and a unifying definition that ignores the shifting dimensions of their various practices will prove to be an insufficient definition. However, as discussed above, the lack of an all-unifying definition is not a defeat for the task of acquiring meaning, but a sign that such meaning must provide an element of openness: an open space through which one may trace the algorithm through the wandering, interconnected and overlapping paths of various disciplines and the complexities that their associated practices may entail.

The algorithm is, first and foremost, a ***step-by-step process of accomplishing a task***: of giving something to an algorithm to transform it into something else. This process of transformation, often obscured in a 'black box' should not be thought of as a mystical process or some form of magic trick that we never see, but an operation that one should – and must – critique if the algorithm is to have any meaning. For the software engineers and developers, an algorithm must be constructed so that the task is able to be completed. This means that there must be no step in the algorithm which contradicts another step, which would cause the algorithm to undertake a calculation that would never end, or creating an infinite loop which it cannot break free of. As has been shown above, the universal language of the computer means that it can be programmed for an unlimited amount of tasks, but this universal language inherently has its limits when it is embedded in the various practices it can be programmed for. With this in mind, algorithms created for engaging in complex tasks with rapidly changing variables and uncertainty should contain methods for finding ways to mitigate the risk of this happening. There can be many ways to accomplish a task, and some will be better than others in different contexts. The engineer/developer must consider the use of the

[27] To be unquestionable implies that it cannot be interrogated; that there is something non-negotiable beyond the rules of Turing's game. For example, in pre-establishing the identity of the human and the computer in such a game, it will also establish the conclusion "thinking belongs to the human, not to the computer", thus locking out the computer from ever gaining the status of thinking. The interrogation game withholds such identities dominating the rules of the game, instead letting the participant's actions speak for themselves.

algorithm to find creative solutions, and to balance efficiency with versatility: if an algorithm has lots of steps to check for a large number of factors, then this will ultimately slow down completion of the task, but if the algorithm only has a few easy steps, then it's quick completion may not take account of variables which could drastically affect the outcome. Understanding what an algorithm is in this instance means understanding how this step-by-step process proceeds from beginning to end, balancing the various needs and complexities that may arise in completing the task.

However, the meaning of "what is an algorithm?" does not belong solely to those who code and program them. Algorithms are used by people with no programming knowledge whatsoever in a multitude of everyday tasks; often without realising that they are interacting with an algorithm. An algorithm may just be a means to an end to those just wanting to accomplish a task in the quickest, efficient, and least disruptive manner possible, and to give the algorithm a meaning and a presence in that process will almost certainly detract from that. Nevertheless, it is important to have at least a general understanding of what the algorithm must do to accomplish its task: as mentioned in the above paragraph, there often exists no single, perfect method to accomplishing a task. Likewise, there is no single solution to how to deal with an algorithm. Where some algorithms require consent from the user to access their data, this may involve an intrusion on their privacy, but in return allows the algorithm to provide a more efficient and personalised service. It is important to a user, who will not wish to become too distracted from the task it has set an algorithm, to have the opportunity to view and interrogate just how their data or interactions will be used by the algorithm and those who employ it. The ever-increasing speed of digital technology and the new frontiers it operates in means that regulation and ethical frameworks for how these algorithms are used often lags behind in the no-so-fast process of government legislation and corporate codes of conduct. Thus it is often up to the users and creators themselves to define and interrogate the limits and meaning of engaging with an algorithm.

It also means something to deal with an algorithm compared to any other type of process, or even a human being. If we delegate a task to an algorithm, we are generally relying on it to accomplish its task in the most efficient, accurate and (ideally) most ethical way possible. If the algorithmic process is obscured inside the black box, or coded in a language we cannot understand, we are at risk of reinforcing the idea of the ephemerality of an algorithm, which evaporates after the task is completed, the finished object leaving no trace of the preceding process. Imagine viewing a work of art and wondering about the process of how it was created: it is easy to attribute the artistic process to the work of 'genius' or a mystical, divine gift. Turing's interrogation game aims to halt such attributions and identities by providing a framework in which a participant's work is judged based on its responses to commands and questions without first establishing between real and artificial,

human and computer etc. The significance of Turing's game is that no such distinction should be created unless the participants create it themselves through their diverging responses. Again, there is no single or ideal way to play this game: it requires creative solutions to keep this distinction from manifesting itself. It is possible that such a distinction establishing itself is unavoidable, but given that the algorithms we interact with are constantly learning and evolving, just as we as humans are learning and evolving our own thought processes, then the possibility of these distinctions and divides materialising can always be stalled and delayed; long enough to fulfil the aim of the game or the task to be completed. If we know or suspect an algorithm is involved and participating in a game or task, then the interrogation method; the questioning of the participants allows us to determine the algorithm's role and just how it is operating.

What is an algorithm? To answer this question one must first cast aside – or at least suspend for a time – the identity of the functional, calculating computer, and demystify the ephemeral process of the algorithm: it is not something inaccessible or fleeting, but rather a tangible, ongoing process that we can critique, question, and interact with. Whether user, engineer or developer, the task for ourselves is to ask the appropriate questions that shed light on the decisions and choices made to accomplish a task; to judge the algorithm's effectiveness, it's consideration for the users data, and so on. In return, we may invest in the algorithm and entrust it with more tasks, providing them with the rich and diverse data it needs to become more useful and efficient. Interacting with the process in this way, one should not seek to stop the algorithm in order to treat it as a static object in order to find its meaning, as this will again cause the step-by-step process to elude this object, and mystifying it once again. Our questioning and critique should take place as the algorithm is active and participating in its task or game, where we will find the answer to the question "what is an algorithm?" as the algorithm participates, configures, and contributes not only to the completion of a task, but how such tasks can be solved and completed in new ways provided by computational processes.

Bibliography

Descartes, René. “To Marsenne, 20 November 1629” in *The Philosophical Writings of Descartes Volume III: The Correspondence*, translated by John Cottingham et al. Cambridge: Cambridge University Press, 1991: 10-13

Davis, Martin. *The Universal Computer: The Road from Leibniz to Turing*. New York/London: W.W. Norton & Company, 2000

Gödel, Kurt. *On Formally Undecidable Propositions of Principia Mathematica and Related Systems*, translated by Brian Meltzer. New York: Dover Publications, 1992.

Leibniz, G.W. “The Art of Discovery” in *Leibniz: Selections*, ed. Philip P. Wiener. New York: Charles Scribner’s Sons, 1951: 50-58

_____. *Explanation of Binary Arithmetic*, translated by Lloyd Strickland, accessed 4th April 2019 http://www.leibniz-translations.com/binary.htm

Seaver, Nick. “Knowing Algorithms”, 2017, accessed 1st May 2019: https://static1.squarespace.com/static/55eb004ee4b0518639d59d9b/t/55ece1bfe4b030b2e8302e1e/1441587647177/seaverMiT8.pdf

Turing, Alan. “On Computable Numbers, with an Application to the Entscheidungsproblem” in *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life: Plus The Secrets of Enigma*, edited by B. Jack Copeland. Oxford: Oxford University Press, 2004: 58-90

_____. “Can Digital Computers Think?” in *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life: Plus The Secrets of Enigma*, edited by B. Jack Copeland. Oxford: Oxford University Press, 2004: 477-486

Whitehead, Albert North, Bertrand Russell. *Principia Mathematica: Volume I*, 2nd edition. Cambridge: University Press, 1968