The Algorithmic Mind
And
What It Means To Solve A Problem

Wim Hordijk

SmartAnalytiX.com
Lausanne, Switzerland

wim@WorldWideWanderings.net

The Algorithmic Mind
And
What It Means To Solve A Problem

In a recent paper in this journal, a claim is made that the mind is not algorithmic. The supporting argument for this claim is that humans frequently solve certain problems which can supposedly not be solved by a computer. However, this argument is based on a fundamental misunderstanding of what it means to solve a problem. Here, I will argue that the provided argument for the claim that the mind is not algorithmic confuses two different meanings of the phrase “to solve a problem”: its formal meaning and its colloquial meaning. As a result, the argument is not logically consistent, and thus does not support the original claim.

Introduction

In a recent paper in this journal, a claim is made that the mind is not algorithmic (Zia et al., 2012, p. 97). The supporting argument for this claim, repeated elsewhere (Kauffman, 2013), is that humans frequently solve certain problems which can supposedly not be solved by a computer. However, as I will try to make clear in this short note, this argument is based on a fundamental misunderstanding of what it means "to solve a problem". In fact, there are two different meanings of this phrase: (i) its formal, theoretical computer science meaning, and (ii) its colloquial meaning. These two meanings are not equivalent, but they are used interchangeably and without distinction in the argument referred to. Therefore, the given argument is not logically consistent, and cannot be accepted as valid.

To explain the difference between the two meanings of the phrase "to solve a problem", it is necessary to explain its formal meaning in a concise and clear way. Below, I will provide a non-technical explanation of this formal meaning. First, the difference between "easy" problems and "hard" problems in computer science is explained. Then the notion of approximate solutions is addressed, which relates to the alternative, or colloquial meaning and the way humans tend to solve problems. Finally, the difference between the two meanings is highlighted. With that in place, I will argue that the original argument provided by Zia et al. (2012) for the claim that the mind is not algorithmic confuses the two meanings of the phrase "to solve a problem", and that it does, therefore, not support the stated claim.

To Solve Or Not To Solve, That's The Question

Easy problems

In theoretical computer science, a distinction is made between "easy" problems and "hard" problems. An example of an easy problem is finding the shortest path between two points in a road network. Every time you turn on the navigation system of your car and type in a destination, it invokes an *algorithm* to compute the shortest route between your current location and your desired destination, using a large data set consisting of all known roads, their junctions, and the distances between junctions. An algorithm is a well-defined, unambiguous, and usually deterministic (although non-deterministic algorithms are also possible) sequence of steps that can be performed on a computer and which will return an answer (the result of the computation) within a finite number of steps. Each step consists of a simple instruction, or an elementary computation, such as comparing or adding two numbers. An algorithm can thus be viewed as the formal equivalent of a recipe for cooking a meal. You start with a given input (the raw ingredients), perform a certain number of well-defined steps (boil 1L of water, cut the onions into small pieces, cook the pasta for 8mins, etc), to end up with the desired result (a tasty meal).

The shortest path algorithm takes as input a given road network, a starting point and a destination, and computes the shortest route between these two points given the provided road network. The nice thing about this particular algorithm is that it can be

shown mathematically that it is an *efficient algorithm*, meaning that it returns the answer within a "reasonable" amount of steps. To determine the efficiency of an algorithm, its *worst-case running time* is calculated. This is the largest number of steps the algorithm might need on any (arbitrary) input, and is expressed as a function of the size of the input. For example, for the shortest path algorithm the input (the road network) can be mathematically described as a *graph*, its *nodes* representing the junctions and its *edges* representing the roads connecting these junctions. For any given graph with n nodes, it can be shown that the shortest path algorithm will never need more than n^2 steps to return the answer. This (worst-case) running time is written as $O(n^2)$. Moreover, it can be proved that the answer it returns is indeed correct, i.e., it always returns the shortest possible path between the starting and ending node, given the input graph. In other words, even though there might be many possible routes between the starting and ending point, the shortest path algorithm always returns the best possible, or *most optimal* solution (the shortest route) for a given road network with n nodes, in no more than n^2 steps.

Now, an algorithm that has a worst-case running time that is *polynomial* in the size of the input is considered to be efficient. So, on inputs of size n, any worst-case running time of, for example, the form $O(n^2)$, $O(n\log n)$, or even $O(n^x)$ for some constant x is efficient: we can expect the answer in an acceptable amount of time[1]. Finally, a given problem is considered "easy" if an algorithm can be constructed for it that will (provably) always return the most optimal solution and that has a worst-case running time that is polynomial in the size of the input. The "easy" problems together make up what is known as problem class **P** (Garey and Johnson, 1979; Papadimitriou, 1993). The *shortest path problem* is in this problem class **P**.

Hard problems

Unfortunately, not all problems encountered in real life are easy. Consider the following extension of the shortest path problem. Instead of wanting to go from point A to B in the shortest possible way, imagine you need to visit several cities and then return to your starting place, but in such a way that the total travel distance is as short as possible. So, the question is in which order to visit the given cities so that the total length of the tour is minimal. This problem is known as the *Traveling Salesman Problem* (TSP), which turns out to be a "hard" problem.

If there are n cities to visit, there are $n!$ (n factorial, which is n times n-1 times n-2,..., times 2 times 1) possible orderings, or tours. This is an exponentially growing number: every time you increase n by one, the number of possible tours more than doubles. Each of these tours has an associated total length (travel distance), and the object is to find the tour with the shortest length. Clearly, just randomly criss-crossing from one city to another is not going to provide the shortest tour. However, so far nobody has been able to construct an algorithm for the TSP problem that will always return the shortest tour and that has a worst-case running time that is polynomial in the number of cities to visit. The only known algorithms that will guarantee to find the shortest tour have a

1 Note that for a very large value of x the worst-case running time might actually be quite large. However, in practice almost all (known) efficient algorithms have a small exponent, usually no larger than 3 or 4.

worst-case running time that is *exponential* in the input size, e.g., $O(2^n)$. In the worst case, we may have to enumerate all the (exponentially many) possible tours to find out which one is the shortest.

To appreciate what this means, assume we have 10 cities to visit. With an exponential running time, we can expect the optimal answer in, roughly, $2^{10}=1024$ steps. That is actually not bad. However, now imagine there are 100 cities. That means the algorithm needs on the order of $2^{100}\approx 1.27\text{x}10^{30}$ steps. Considering that the age of the universe is estimated to be about $4.4\text{x}10^{17}$ seconds, even if we could perform thousands of computing steps in one second, the lifetime of the universe would still be several orders of magnitude too short to wait for the optimal answer! Given that modern-day networks (such as mobile communication networks or high-performance computer chips) easily have thousands of nodes, there is no hope to find the most optimal solution for TSP or equivalently hard problems on such networks.

Problems for which the only known algorithms to always find the most optimal solution all have a worst-case running time that is exponential in the size of the input, are (appropriately) called *intractable* – we may have to wait "forever" to get an answer. Intractable problems for which it is still easy to *verify* a solution (if one was somehow provided), together with the problems already in **P**, make up the larger problem class **NP** (for nondeterministic polynomial). Moreover, for some of these intractable problems it can be shown mathematically that they are at least as hard as any other problem in **NP**. These "hardest" problems are called *NP-complete* (Garey and Johnson, 1979; Papadimitriou, 1993). TSP is such an NP-complete problem.

Note that it is not only the fact that there are an exponential number of potential solutions that make these problems hard. For example, in principle there could also be an exponential number of possible paths between any two given points in a road network. However, for the shortest path problem it is possible to construct an efficient algorithm to find the optimal solution. For the TSP problem this is not possible, or at least nobody has been able to find one so far (although many very smart computer scientists and mathematicians have tried).

As a technical aside, theoretically it is not known whether it is, in principle, impossible to construct an efficient (polynomial-time) algorithm for NP-complete problems. In other words, it is still an open problem in theoretical computer science whether **NP** is strictly larger than **P**, or whether they are actually equal. From the practical evidence so far, the former seems more likely, but mathematicians are still trying to resolve this question formally, one way or the other.

Approximate solutions

Unfortunately, it turns out that there are many real-world optimization problems that are NP-complete. Such problems occur for example in engineering, economics, logistics, communication networks, and also in science, to name just a few. However, for many of these problems it is not always necessary to find the *most optimal* solution. In many cases, an *approximate* (sub-optimal) solution suffices.

This happens in nature as well. The human brain continuously comes up with approximate solutions to the problems we face in daily life. We may not always be able to calculate the absolute shortest path from A to B, but as long as we get where we want to be without too much delay, things are fine. Evolution can also be viewed as an optimization process, generating solutions (different species) to the problem of survival and reproduction, which is a basic requirement for all of life. However, most of these solutions are not perfect or optimal. But in evolution, any trait (whether physiological or behavioral) that provides an increase in an organism's chances for survival and reproduction, will have a selective advantage.

Obviously computers can also be used to find approximate solutions to hard problems. Even if a problem is intractable, or NP-complete, it is often still possible to construct efficient algorithms (with a polynomial worst-case running time) that return approximate solutions which, although not necessarily the most optimal, are usually “good enough” for practical purposes. Neural networks and genetic algorithms are examples of such approximation algorithms. A neural network (Lippmann, 1987; Haykin, 1999) is a computer algorithm that mimics the workings of the brain, and that can be trained to, for example, recognize patterns such as your fingerprints. Just as our own brain, it may not always perform flawlessly, but it can achieve a high level of accuracy which, for most practical applications, is acceptable. Similarly, a genetic algorithm (Holland, 1975; Goldberg, 1989, Mitchell, 1996) tries to find approximate solutions to difficult problems by mimicking an evolutionary process, literally evolving better and better solutions over time. In practice, these approximate solutions are often adequate enough.

Solving problems

So what does it mean to solve a problem? As the above explanation of hard problems and approximate solutions already indicates, there are actually two meanings of the phrase “to solve a problem”. The first, a formal definition from theoretical computer science, is “to find the *most optimal* solution out of all possible ones to a given problem”. As discussed, in the context of this formal meaning it can be shown that there exist problems which are *intractable*, or in other words *unsolvable* (at least in practice), even for a very fast computer. And these intractable (NP-complete) problems are not just some rare theoretical construct, but they appear in many situations in daily life.

The second, colloquial meaning is “to find an *approximate* solution that, given the circumstances, is adequate enough”. Such an approximate solution might not even be close to the optimal one, but given that we have a limited amount of time and resources available, we will have to be satisfied with it. This is the way nature and the human brain “solve problems”, which can also be mimicked or simulated by a computer, in a purely algorithmic way.

The Algorithmic Mind

The argument used by Zia et al. (2012) to claim that the mind is not algorithmic, correctly states that there exist problems that cannot be solved algorithmically

(efficiently, that is, in the formal, theoretical computer science meaning of the phrase “to solve”). In fact, they make an even stronger argument that there are certain types of problems, such as the *frame problem*, which cannot be solved algorithmically *in principle*, even if we would allow for an exponential amount of computing time (since the number of possible solutions can potentially be innumerable). In artificial intelligence, the frame problem refers to listing all relevant features, variables, and their possible relations necessary to solve a given problem (i.e., it is a meta-problem). However, the main idea is the same: no computer or algorithm can *efficiently* find the *most optimal* solution to these hard problems. Agreed.

But then they go on to say: “Yet people solve such problems all the time. This is, we claim, a powerful line of evidence that the human mind is not algorithmic.” (Zia et al., 2012, p. 97). Disagreed! Somehow, the authors seem to have switched to the colloquial meaning of the phrase “to solve” in the second part of their argument. In fact, the argument is preceded by an anecdote where one of the authors claims to have solved the frame problem in a particular real-life situation. However, it is clear that his claimed “solution” is only an approximate one, not necessarily (and unlikely) the most optimal one. Humans rarely solve problems in the formal, theoretical computer science sense. If we did, we would not need computers, or at least we would beat them at playing chess. However, we indeed solve problems in the colloquial, approximate way all the time. But, as discussed above, there do exist (efficient) algorithms, such as neural networks and genetic algorithms, that can also solve these problems in very similar (approximate) ways as humans do (and often they find even better solutions than human engineers).

In short, the argument of Zia et al. (2012) confuses the two meanings of the phrase “to solve a problem”: the formal meaning (used in the first part of the argument) and the colloquial meaning (used in the second part of the argument). Thus, their “powerful line of evidence” is based on a misunderstanding and confusion of what it means “to solve a problem”, and their claim that the human mind is not algorithmic is therefore not supported by it.

Such a misunderstanding when there are multiple meanings of a particular phrase, a formal one and a colloquial one, unfortunately happens more often (see e.g. Szathmáry (2013) for an example in biochemistry). Of course this does not necessarily mean that the mind *is* algorithmic. For example, Kauffman (2013) also invokes evidence from new developments in quantum mechanics to argue that the mind is not algorithmic. I will have to leave it up to other, more knowledgeable scientists on that topic, to judge the validity of these alternative arguments. However, with this brief note I hope to have argued adequately that the provided computational argument for a (possibly) non-algorithmic mind is invalid.

References

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A guide to the Theory of NP-Completeness,* W. H. Freeman.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation,* Prentice Hall.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems,* University of Michigan Press.

Kauffman, S. (2013). “Answering Descartes: Beyond Turing,” in S. B. Cooper and A. Hodges (eds.), *The Once and Future Turing: Computing the World*, Cambridge University Press.

Lippmann, R. P. (1987). “An introduction to computing with neural nets,” *IEEE ASSP Magazine* 4:4–22.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms,* MIT Press.

Papadimitriou, C. H. (1993). *Computational Complexity*, Addison-Wesley.

Szathmáry, E. (2013). “On the propagation of a conceptual error concerning hypercycles and cooperation,” *Journal of Systems Chemistry* 4:1.

Zia, A., Kauffman, S., and Niiranen, S. (2012). “The prospects and limits of algorithms in simulating creative decision making,” *E:CO* 14(3):89–109.

Acknowledgements

I would like to thank Stuart Kauffman for many inspiring discussions, even if we do not always agree. Thanks also go to my colleague Mike Steel for helpful comments on an earlier version of this note.