CS 254 Final Project

March 15, 2019

1 Introduction

As the concept of computability has its roots in philosophy, countless scholars have closely studied computability in CS through the lens of philosophical logic, and vice versa. However, as Scott Aaronson asserted in his essay entitled "Why Philosophers Should Care About Computational Complexity" [1], for many philosophical questions, complexity theory offers many more insightful perspectives than computability alone.

Indeed, many of the most basic questions about the meaning of life are technically computable – just not efficiently computable, nor close to it. Consider for example: how did intelligence begin on Earth? The answer to this question is theoretically computable – it happened through laws of physics that could potentially be simulated with enough time and computational power – but as Gödel argued in a letter to the logician Hao Wang, the evolution of intelligent life seems exponentially unlikely to happen on mere geological timescales. Gödel acknowledges that this is computable; further, natural selection makes things happen faster than random chance, and there is still much to learn about precisely how this occurs. If complexity theory can tell us "exactly how complicated" this creation of intelligent life is, that does not instantly answer the question – but that could tell us a lot about how lucky we were that such life developed on Earth, and how likely it is to occur on other planets.

In my paper, I will turn my attention to just two topics that especially stuck out to me in Aaronson's paper (it was long, so I can't summarize it all here with any reasonable level of depth). In particular, I will focus on some cool connections of complexity theory to time travel and non-biological sentience.

2 Does causally consistent time travel imply P = NP?

In computer science, we utilize time and space differently; importantly, space can be reused, while time cannot. This natural intuition follows from our everyday life experience. We always progress forwards in time at a relatively constant rate, and cannot go back to reuse an afternoon that we did not utilize as efficiently as desired. By contrast, we can go back and reuse the same physical space over and over; we do this all the time as we move about the world during the day and return to the same places in our homes every evening. But what if we allow time travel?

One seemingly obvious idea to solving an NP problem in polytime would be to use up an exceedingly large amount of time, then travel back in time to before the computation

began. Indeed, this would produce an answer within polytime; however, this could still take exponential resources, as the machine is busy for more than polytime. Furthermore, if the amount of time needed to compute something is longer than the existence of the Universe, this becomes incredibly impractical to say the least. Thus, I will define P in the presence of time travel to only be able to use polynomial resources in total.

2.1 Closed Timelike Curves

The notion of time travel that I will work with in this paper is as follows:

Definition 2.1. A Closed Timelike Curve (CTC) is a path along space-time on which time always moves forward like normal, but then ends back where it started.

This feels like something that couldn't possibly exist, but Einstein's laws of general relativity allow for this to happen in theory. Naturally, a lot of people have debated whether or not this is possible in real life just because it is mathematically allowed. However, other seemingly impossible phenomena, like black holes, were predicted by mathematics and theoretical physics well before being observed in real life. So I will not discount this possibility immediately.

One paradox that often gets cited with regards to CTCs is the grandfather paradox. Consider an individual, Alice, who went back in time and while there, killed her grandfather, Bob, before Bob had any children. If Bob has no children, Alice never existed. But if Alice never existed, how could Bob have died? Thus we've reached a contradiction, and Alice therefore cannot go back in time to kill Bob.

This contradiction does not immediately imply though that CTCs do not exist; it just means that we will want to have *causally consistent* CTCs. Otherwise, we get paradoxes like this.

One possible solution to making the grandfather paradox causally consistent is having nature prevent such events somehow, for example, by having the gun jam right at the necessary moment. Admittedly, this feels kind of sketchy. Fortunately, Deutsch proposed another solution in his paper, "Quantum mechanics near closed timelike lines" [6]. In this paper, he describes how all CTCs must mathematically have at least one *fixed point* as long as the laws of physics are indeed quantum-mechanical; and that this implies that different possible states around the rest of the loop happen with different probabilities (in this case, Alice is born and goes back to shoot Bob with probability 1/2). We will show how to resolve this more formally in the next section.

Deutsch's theory still receives its share of criticism. For example, Deutsch demonstrates that a CTC has at least one fixed point: if it has more than one, which solution will Nature choose? People have also done various simulations with putting one of two entangled particles into a CTC and reaching a contradiction with other theories. But consider the following more simple problem: what if finding the fixed point on the CTC is really difficult, astronomically difficult in fact? This seems very likely, especially given how complicated the physical world is. This is where complexity theory becomes essential.

2.2 How to solve NP-complete problems with polynomial resources

Definition 2.2. Computation of a function $C(w)$ on a CTC acts like a normal deterministic Turing Machine with the following extra properties:

- The algorithm is allowed to specify an efficient function $f : X \to X$, where $|X|$ is finite and $f(x)$ runs in polytime in $|w|$ for all $x \in X$.
- Nature writes down a Markov chain M_f where each element of X is a node and each edge is $(x, f(x))$ for $x \in X$. Then nature (adversarially) chooses a stationary distribution D of M_f, then picks a random sample y from D. The algorithm may then use y in its computation.

To better motivate this definition, let us walk through the example of the grandfather paradox as described in the previous section. For notational convenience, define $x_1 :=$ "Bob gets killed before having any children" and $x_2 :=$ "Alice does not kill Bob". Then in this scenario,

$$f(x_1) = x_2$$

and

$$f(x_2) = x_1$$

because as defined in the scenario, if Alice is born, she definitely kills Bob, but if Bob is killed before having children, then Alice cannot be born. Thus, the Markov chain consists of two nodes: x_1 and x_2, and there are two directed edges each with weight 1: $(x_1, x_2), (x_2, x_1)$. For this Markov chain, the only stationary distribution is

$$\begin{cases} x_1 \text{ with probability } .5 \\ x_2 \text{ with probability } .5. \end{cases}$$

Then, nature picks either x_1 or x_2 with their respective probabilities (in this case, .5 each), and reality proceeds from there.

Theorem 1. NP = $\mathrm{P}_{\mathrm{CTC}}$, where $\mathrm{P}_{\mathrm{CTC}}$:= the set of problems that can be solved with polynomial resources.

Proof. Consider the following NP-complete problem: given a circuit computing some function

$$f : \{0, 1, ..., 2^n - 1\} \to \{0, 1\},$$

return whether there exists an element x of $\{0, 1, ..., 2^n - 1\}$ such that $f(x) = 1$. If we do the following algorithm within a CTC, we can actually get a solution in polytime. First we define an intermediary computable function $g : \{0, 1, ..., 2^n - 1\} \to \{0, 1, ..., 2^n - 1\}$:

def g :
    Given input $x \in \{0, ..., 2^n - 1\}$:
        if $f(x) = 1$, return x
        else, return $x + 1 \bmod 2^n - 1$

Then we define $C(f)$ as follows:

def $C(f)$:

construct g
nature gives you y from stationary distribution D of M_g
return $f(y)$

To show this algorithm works on a CTC computer, we first calculate the Markov chain for this g. Note that for each x, $g(x)$ is deterministic, so all edges have weight 1.

- If there is no satisfying assignment to f, then $\forall x,\ g(x) = x + 1$. Thus the only stable distribution for this Markov chain is to have all the nodes with probability $1/2^n$ for each. Then nature returns a random y. And in this case, $f(y) = 0$ for all y. Then no matter which node nature returns, $f(y) = 0$. Thus the algorithm returns 0.
- If $\exists x$ such that $f(x) = x$, then the only stable distributions for this Markov chain are supported on nodes x' whose outward edge is (x', x') — that is, subsets of $\{x : f(x) = x\}$. Then nature must return $y \in \{x : f(x) = x\}$, which implies that the algorithm returns $f(y) = 1$.

Thus the algorithm returns 1 iff there exists y such that $f(y) = 1$.

■

2.3 PSPACE $\subseteq$ $\mathbf{P_{CTC}}$.

As you might suspect from the above section, $\mathrm{P_{CTC}}$ is significantly more powerful than plain P (if we believe that P $\neq$ PSPACE). Indeed, Aaronson and Watrous[3] proved that polynomial resources in a CTC can solve exactly those problems in PSPACE. I will not prove here that $\mathrm{P_{CTC}} \subseteq$ PSPACE (though it is a true fact), but I will sketch a proof for the other direction:

Theorem 2. PSPACE $\subseteq \mathrm{P_{CTC}}$.

Proof. Let M be a deterministic polyspace Turing Machine deciding some language L, and call $\{m_i\}$ the sequence of configurations of M when run on some string x. Also assume that M always terminates from any configuration, including those that might not be reachable from any string input.

For m a state of M and $b \in \{0, 1\}$, consider the function

$$f(\langle m, b\rangle) = \begin{cases} \langle m_1, 1\rangle & \text{if} \quad m \text{ is in "ACCEPT" state} \\ \langle m_1, 0\rangle & \text{if} \quad m_i \text{ is in "REJECT" state} \\ \langle m', b\rangle & \text{else} \end{cases}$$

where m' is the configuration reached by running M one more step.

Note that since M decides L, there must exist some terminal state m_T that is "ACCEPT" or "REJECT". Then consider giving f to a CTC computer. A part of the Markov chain for f is

$\langle m_1, 0\rangle \to \langle m_2, 0\rangle \to ... \to \langle m_T, 0\rangle \to \langle m_1, \tilde{b}\rangle$ and $\langle m_1, 1\rangle \to \langle m_2, 1\rangle \to ... \to \langle m_T, 1\rangle \to \langle m_1, \tilde{b}\rangle$

where

$$\tilde{b} = \begin{cases} 0 & \text{if} \quad m_T = \text{“REJECT” state} \\ 1 & \text{if} \quad m_T = \text{“ACCEPT” state.} \end{cases}$$

There are also other nodes in the chain, but mapping f under them will always eventually reach $\langle m_1, 1\rangle$ or $\langle m_1, 0\rangle$ since M always terminates. Note that the only stable distribution is the uniform distribution over the set of nodes

$$\langle m_1, \tilde{b}\rangle, \langle m_2, \tilde{b}\rangle, ..., \langle m_T, 0\rangle.$$

Thus, nature gives you $y = \langle m_i, \tilde{b}\rangle$ for some $i \in \{1, ..., T\}$. From that, our algorithm simply returns $\tilde{b}$ and we're done.

■

2.4 Final thoughts on other weird consequences of causally consistent time travel

In a sense, CTCs makes time and space equivalent, not by doing the naive solution of doing a computation in a lot of time then going back to reuse the time, but by allowing time to be reused. Interestingly, Aaronson also noted that quantum computers inside the CTC don't help solve more difficult problems, but I will not go into detail about that here.

In summary, there are lots of fun puzzles to consider with time travel, and complexity theory gives us a new way to formalize, and potentially solve, these problems.

3 Can we quantify a soul? Can a computer have a soul?

As defined by the Oxford English Dictionary, a “soul” is:

Definition 3.1. Soul: The principle of intelligence, thought, or action in a person (or occasionally an animal), typically regarded as an entity distinct from the body; the essential, immaterial, or spiritual part of a person or animal, as opposed to the physical.

This feels like an unwieldy definition to handle when trying to determine whether or not machines can possess such a thing. How do we formally characterize the ways in which humans have consciousness that machines don't possess? Can a machine fall in love, write a heartfelt song, experience grief? These questions become more and more relevant as engineers design increasingly “smarter” machines. Can AI ever pose a threat to humanity if it becomes more powerful? Is there any threat of AI if AI can never become conscious? What does it mean for AI to become conscious?

Around the 1950s, Turing formalized a way to approach this question. He separated the matter of having a soul into two different questions: is it possible for a machine to pass a test such that the machine appears in all ways to be a sentient life; and if a machine passes such a test, would it then be right to attribute personhood to it? In this half of my paper, I will discuss both of these questions set out by Turing, beginning with his famous Turing Test.

3.1 The Turing Test

Definition 3.2. The Turing Test is where a human tester chats with some people and some computer bots through a chat box. The goal of the program-writers of the computers is to have their computer program pass off as a human from the perspective of the tester; the tester tries to figure out which chat members are humans and which are bots.

At the time he designed this test, Turing imagined that easily by the year 2000, machines could pass the Turing Test. In actuality, this has proved to be much more difficult; although there have been countless great strides in computer science, especially within machine learning, no machine can talk fully human-like yet. There have been some robots who are somewhat successful, but in certain conditions.

Eliza [8] is one of the earliest examples of an AI that seemed reasonably human-like. Her design was primarily to try to have the other person do most of the talking, something that many real humans do, which also reduces the amount of human-like dialogue she would need to say. Another AI, Parry [5], was designed differently; his program aimed to constantly steer the conversation towards a few topics he knew well, and about which his persona was "paranoid." There have been still more recent attempts, such as Cleverbot [4], whose design is to use a lot of databases of human conversation to compose responses – but the lack of a consistent personality reveals Cleverbot's lack of humanity. One of the most successful attempts at simulating human conversation was Eugene Goostman [2], an AI pretending to be a 13-year-old Ukrainian boy. Indeed, 33% of the testers at the Turing test competition at the Royal Society in 2014 believed that he was sentient. However, when Scott Aaronson purposely messed with Eugene, this is the result:

"Scott: Which is bigger, a shoebox or Mount Everest?

"Eugene: I can't make a choice right now. I should think it out later. And I forgot to ask you where you are from...

"Scott: How many legs does a camel have?

"Eugene: Something between 2 and 4. Maybe, three? :-))) By the way, I still don't know your specialty – or, possibly, I've missed it?" [2]

This conversation continues awhile longer; I would highly recommend reading it for entertainment. However, the opening lines suffices to demonstrate Eugene's lacking of human intelligence. Instead, his persona's young age and cultural differences are both features that help convince people to not judge his mistakes as badly – although when Aaronson purposely asks ridiculous questions, Eugene's strategy is exposed.

Something all of these machines have in common is that they imitate elements of being a human other than intelligence. Thus, we are still not very close to having a machine fully pass as a human.

3.2 How complexity theory helps with strengthened Turing Tests

Observe that it takes a short finite amount of time to decide that someone is sentient. Thus, theoretically, we could have a finite – albeit very big – look-up table defined:

$$T : \text{what other people say} \mapsto \text{what you should say in response.}$$

Using T, we could simulate sentience as follows:

while(conversation continues):
    x = tester's input
    reply $T(x)$

The above algorithm would simulate sentience, which implies that the task of imitating sentience is computable (as long as we assume that deciding if someone is sentient takes a finite amount of time). However, no one has managed to make a robot that accesses such a table yet because making the table is a very inefficient task. Thus, when considering if it is possible to make a robot act like a human, we must not just consider if doing so is theoretically possible, but also, if running the robot is efficient or not. We already know that a theoretical model of such a robot exists by the above approach of using a look-up table. However, there are two key problems: (1) the look-up table would likely be infeasibly large; (2) the look-up table would take astronomical time to create, if we even know how to create it.

As a related side note: Turing Tests are a form of interactive proofs – are these fundamentally different than other styles of proofs with respect to the information they are able to qualitatively give us about sentience? For example, what if we do not allow full interactivity, but only a Merlin-Arthur protocol? (Answer: humans are crude. [7]. But passing as human with only one word seems easy enough to simulate with a computer program, at least as well as a human can.) Would some fundamentally different formulation of a Turing Test give us qualitatively different information about what makes someone conscious? These are other questions worth pondering, as perhaps some other proof format could reveal sentience more efficiently.

3.3 What do humans do that machines (currently) don't?

Even if we could have a computer correctly answer questions about size or number of legs on a camel, what about other human characteristics besides intelligence? It would be nice to create a complexity class of types of problems that humans can solve, whether it be simple arithmetic, writing an original poem, or feeling happy. Simple arithmetic is within P, and so it seems that P $\subseteq$ human-solvable problems. However, if P $\neq$ NP, then intuitively it does not seem that humans can solve everything in NP or coNP efficiently: for example, factoring big numbers is hard.

Aaronson proposes two possible scenarios: things humans can do better are not comparable to NP problems, or humans are good at special cases of NP. These each raise some follow-up questions. If things that humans are good at are not comparable to NP problems, then how would we classify the complexity class of human-solvable problems from a computational standpoint? What would be a good example of a human-complete problem? Or if humans

are good at special cases of NP problems, would we eventually be able to have computers solve these special cases efficiently? Assuming P $\neq$ NP, would human-complete problems be in P, NP\P, or outside of NP?

Note that if we can formally define a task that efficient computers cannot do, then that task can be used as a Turing test that no efficient computer can ever pass.

3.4 Characterizing human intuition

So far we have discussed what types of problems humans can solve. But on an even more basic level, how could a machine parallel the concept of what it means for a human to "know" something?

Usually modern analytic philosophy assumes that if we know something, we know all logical conclusions from that thing. For instance, if we know the number 7, and we also know addition, we also know $7 + 7 = 14$. This concept is called "logical omniscience." However, in real life, logical omniscience has complications. Despite the fact that I know the rules of chess well – my little brother is a national chess master – I certainly couldn't tell you what the next best move is in a chess game. I don't know how to prove Fermat's Last Theorem, despite knowing how to write math proofs. Thus, it seems that for the concept of consciousness, we are more interested in what can be efficiently computed with a human brain. Fortunately, this also lines up well with the concept of "knowledge" in mathematics: we say that the largest prime number "known" is $2^{82589933} - 1$, even though the prime number $p_0 = \min\{p : p \text{ is prime and } p > 2^{82589933} - 1\}$ is well-defined.

Thus, perhaps we define human "knowledge" to be that which is computationally feasible with a human brain. But beyond that which we can compute, humans often also have "feelings" or "intuitions" for things. For example, P $\neq$ NP seems like something that must be the case, even though no one has managed to prove that conjecture yet. Is there a way we can quantify these intuitions and have a machine simulate them? Thus, we must broaden our definition of a machine working like a human brain to include these feelings. Until we can formally quantify "knowing" things in a reasonable way, it not even clear what we would be trying to have a computer accomplish.

4 Conclusion

In conclusion, the same way that computability and philosophy are closely intertwined, complexity theory and philosophy have much deeper connections that we can explore. From time travel to artificial intelligence, there are countless philosophical questions that can utilize complexity to formalize the question.

References

[1] Scott Aaronson. Why philosophers should care about computational complexity. *Computability: Turing, Gödel, Church, and Beyond*, pages 261–328, 2013.

[2] Scott Aaronson. My conversation with "eugene goostman" the chatbot that's all over the news for allegedly passing the turing test. *The Blog of Scott Aaronson*, 2014.

[3] Scott Aaronson and John Watrous. Closed timelike curves make quantum and classical computing equivalent. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 465(2102):631–647, 2008.

[4] Rollo Carpenter. Cleverbot, 2011.

[5] Kenneth Mark Colby. Modeling a paranoid mind. *Behavioral and Brain Sciences*, 4(4):515–534, 1981.

[6] David Deutsch. Quantum mechanics near closed timelike lines. *Physical Review D*, 44(10):3197, 1991.

[7] John P McCoy and Tomer D. Ullman. A minimal turing test. *preprint*, 2018.

[8] Joseph Weizenbaum et al. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.