THE BUSY BEAVER PROBLEM

ANANT ASTHANA

Abstract. In this paper, we introduce the intuition and technicalities behind the Busy Beaver problem, after which we discuss the implications of Busy Beaver results to incompleteness. Subsequently, we mention some interesting properties of the Busy Beaver sequence (namely that it is uncomputable). Finally, we briefly discuss the Ackermann function as well as the inverse Ackermann function.

1. The Intuition

The Theory of Computation is led to the development of the modern-day field of computer science. Here, we investigate an interesting problem regarding an ancestor of computers known as *Turing machines* (TMs); in essence, a Turing machine can read in input and, for our purposes, write on top of and erase the input, as well as navigate only along the tape containing the input. Thus, in a sense, the TM is limited to 1 infinitely long tape extending in both directions.

With these restrictions in mind, we ask the following question:

Question 1. *Given a tape with a bunch of* 0*s initially on it, what is the maximum running time of any* n*-state halting TM on that input?*

This is a very interesting question that has kept researchers coming back to it for decades. Of course, there are a number of variants of this problem, as different metrics for the running time of a TM have been devised, including counting the number of 1s written on the tape rather than the runtime. However, for the purposes of this paper, we will consider the question proposed above.

Date: December 12, 2022.
Thanks to Mr. Simon Rubenstein-Salzedo and the Euler Circle institution.

2. Background and Definitions

Definition 1. A **Turing machine** is a septuple $(Q, q_o, F, \Gamma, \square, \Sigma, \delta)$ where

- Q is the set of states
- $q_o \in Q$ is the starting state
- $F \subset Q$ is the set of accepting states
- Γ is the tape alphabet
- $\square \in \Gamma$ is the blank symbol
- $\Sigma \subset \Gamma$ is the input alphabet
- and $\delta : Q \times \Gamma \times \{L, R\} \to Q$ is the transition function, where at each step, we choose to move the head to either the left or right on the tape.

However, this is a formalized version of the intuitive construct proposed in the first section.

For the sake of the Busy Beaver problem, we assume all TMs to be 1-taped, although variants such as multitaped TMs or 2-dimensional taped TMs have been researched.

Typically, we define Turing machines in terms of an algorithm that governs their transition function δ. As a result, Turing machines can be thought of as simplified expressions of a computer.

Note that the transition state function for the Turing machines considered in the Busy Beaver problem take a specific form. Namely, that form is $a \to bCD$ or $a \to$ HALT, where

(1) a is the symbol read in the current cell on the input tape
(2) b is the symbol to write in place of the current symbol
(3) C is the next state to move to, and
(4) D is L or R, dictating whether the TM head should move to the left or right on the tape.

Thus, for a given state of input, depending on the symbol in the current input tape cell, there are $4n+1$ possible instructions. Since the symbol in the current cell could be 0 or 1, we have $(4n+1)^2$ total possible instructions for one state, leading to $(4n+1)^{2n}$ total algorithms on n states.

Definition 2. The runtime of a Turing machine M is denoted by $s(M)$. For Turing machines that do not halt, we assign $s(M) := \infty$.

Definition 3. Let $T(n)$ denote the set of Turing machines on n states. Therefore, $|T(n)| = (4n+1)^{2n}$. Then, we define $S : \mathbb{N} \to \mathbb{N}$ such that

$$S(n) = \max_{M \in T(n), s(M) \neq \infty} s(M).$$

In other words, $S(n)$ is the maximum runtime any n-state halting Turing machine can have.

Current State	a	b	C	D
q_o	0	1	q_1	R
q_o	1	1	q_1	L
q_1	0	1	q_o	L
q_1	1	HALT	HALT	HALT

FIGURE 1. A sample transition function that encodes the algorithm of a 2nd Busy Beaver machine.

3. The Problem

The *Busy Beaver* problem asks the following question:

Question 2. *Across all halting* 1*-tape TMs with* $\Sigma = \{0, 1\}$*,* $\Gamma = \{0, 1, \square\}$*, and* $|Q| = n$ *what is the maximum runtime on the input* $\ldots 000 \ldots$*?*

Note that the question above considers only halting TMs. Thus, the maximum runtime is not allowed to be indefinitely large.

We can also rephrase the Busy Beaver problem in shorter terms:

Question 3. *For a given* n*, what is the value of* $S(n)$*?*

Remark 1. It's curious that the solutions to the Busy Beaver numbers are denoted by the function $S(n)$. This is due to the idea that Radó did not originally refer to the problem as the Busy Beaver problem, but rather as the *shifting problem* (since the Turing machine is essentially shifting back and forth across the tape). Hence, we use the letter S to denote the Busy Beaver numbers.

Definition 4. Across all halting n-state Turing machines, a TM that with the greatest runtime is known as an nth Busy Beaver Turing machine. In other words, if a Turing machine $M \in T(n)$ satisfies $S(n) = s(M)$, then M is considered an nth Busy Beaver Turing machine.

For example, $S(1) = 1$, $S(2) = 6$, $S(3) = 21$, $S(4) = 107$, $S(5) \geq 4.7 \cdot 10^7$, and $S(6) \geq 8.6 \cdot 10^{15}$ [3]. As one may observe, the maximum runtimes for n-state TMs increase quite quickly as n increases, even faster than exponentially.

Definition 5. We define the class of Turing machines

$$\text{BB} = \{M : M \text{ is an } n\text{th Busy Beaver machine for some } n \in \mathbb{N}\}.$$

For instance, the transition function in Figure 1 details the transition function for a 2nd Busy Beaver.

However, determining if a TM is a Busy Beaver is not a simple passtime hobby. Rather, the Busy Beaver problem has practical implications when it comes to pushing the boundary of our current understanding of mathematics.

4. Beavers and Mathematical Intelligence

Besides increasing popular support for environmentalists, the Busy Beaver problem also allows us to consider the boundary of what is knowable within a sound axiomatic system (in other words, an axiomatic system within which provable statements are true). Let's begin by taking a look at the following statement:

Theorem 4.1. *Let $\mathcal{S}$ be a sound (thus, consistent) axiomatic system such that the axioms and rules of $\mathcal{S}$ can be expressed in b bits of information. Then, there exists a constant $n_{\mathcal{S}}$ so that any statement of the form*

$$S(n) = k$$

where $n, k \in \mathbb{N}$ and $n > n_{\mathcal{S}}$ is unprovable in $\mathcal{S}$.

The proof of this theorem warrants mentioning, as it provides the foundation for the connection between the Busy Beaver problem and a number of currently unsolved problems.

Proof. Say we have a Turing machine $M \in T(n_{\mathcal{S}})$ that does the following: given an input with all 0s, M runs all possible proofs derived from the axioms of $\mathcal{S}$. If it ever finds a proof that $1 = 0$, then it halts. Of course, we know M will never halt. However, notice that $\mathcal{S}$ itself does not have the necessary axioms to prove that M never halts, as this would imply S can prove its own consistency, violating Gödel's Second Incompleteness Theorem.

Now, consider proving $S(n) = k$ for some $n > n_{\mathcal{S}}$. Assume for the sake of contradiction that it is possible to prove a statement such as the one above. However, this would mean that $\mathcal{S}$ can prove M never halts by simulating M for up to k steps and confirming that M does indeed fail to halt. This would violate Gödel's Second Incompleteness Theorem, as above. Therefore, it is impossible to prove that $S(n) = k$ for $n > n_{\mathcal{S}}$ and $k \in \mathbb{N}$. □

Bhaskarpillai and Chaitin [4] have noted that the value of this constant is actually $n_{\mathcal{S}} = b + O(1)$. Therefore, we now have an upper bound on the values of $S(n)$ that can be calculated in a specific set of axioms.

For instance, consider the Zermelo-Fraenkel set theory ($\mathcal{ZF}$), in which all constructs used in modern-day mathematics can be defined. Stefan O'Rear [5], in 2016, proved that

Theorem 4.2. *There exists a 748-state TM that halts if and only if $\mathcal{ZF}$ is inconsistent.*

Corollary 1. *$\mathcal{ZF}$ does not have the axiom foundation to prove the value of $S(748)$, according to Gödel's Second Incompleteness Theorem. Therefore, $n_{\mathcal{ZF}} = 748$.*

The theorem above is a clear example of how the Busy Beaver problem provides insight into the fundamental axioms on which our system of mathematics is built.

Take the Reimann Hypothesis, for example. Mathematicians have been unable to prove or disprove it. Yet, the answer lies within the reach of a 744-state Busy Beaver machine [6].

In fact, there are a number of unsolved problems that can be expressed in terms of the Busy Beaver Problem. Similar cases have been found for the Goldbach and Collatz Conjectures: an anonymous Github user [7] created a 27th Busy Beaver machine that halts if and only if the Goldbach conjecture is false, and Pascal Michel

showed that the current 5-state TM with maximum runtime "exhibits behavior similar to that of the function described in the Collatz conjecture" [2, 8].

What does this mean when it comes to solving unresolved problems? Well, if some clever mathematician constructs a z-state TM (where $z \in \mathbb{N}$) that halts if and only if Zermelo-Fraenkel set theory is inconsistent, and $z < 744$, then it would be impossible to tell if the Reimann hypothesis is true. This is likely the case, as Aaronson believes that $n_{\mathcal{ZF}}$ is somewhere around 20 [2]. Likewise, if $z < 27$, then Goldbach's conjecture would lie in the hands of some extremely clever mathematician. Finally, if z were to hypothetically equal 5, then the Collatz Conjecture would likely remain a mystery.

5. The Cool Uncomputability of BB

Here, we discuss arguably the most interesting property of the Busy Beaver problem.

Firstly, we will show the following basic property of the Busy Beaver numbers:

Theorem 5.1. *For any $n \in \mathbb{N}$, $S(n) \leq S(n+1)$.*

Proof. A proof proposed by the author is as follows: say M is a nth Busy Beaver TM. Then, let M_2 be the TM with $n+1$ states. However, we can set $\delta_2 = \delta$. Thus, $s(M) = S(n) = s(M_2)$. In addition, we know that $s(M_2) \leq \max_{M' \in T(n+1), s(M') \neq \infty} s(M') = S(n+1)$. Thus, $S(n) \leq S(n+1)$. □

However, as noted in the previous section, it's clear from the values of the first few Busy Beaver numbers that the sequence grows very large very quickly. Here, we look at just how fast they actually grow.

Theorem 5.2. *The function S is uncomputable.*

Proof. Say we have a function $f : \mathbb{N} \to \mathbb{N}$ so that $f(n) \geq S(n)$ for all n. Assume for the sake of contradiction that $f(n)$ is computable. Then, we construct a Turing machine T to decide if a TM M with n states halts in the following way:

(1) Read in $\langle M \rangle$.
(2) Simulate M on the input consisting of all 0s, and check the state of the TM after $f(n)$ moves. If M has halted, then accept. If M has not halted, then reject, since we know that the maximum runtime for any halting TM with n states is $S(n)$.

However, since the Halting problem is undecidable, there cannot exist a TM T that can compute $f(n)$. Therefore, $S(n)$ is uncomputable.

□

A direct result of this theorem is the following:

Corollary 2. BB *is undecidable.*

In other words, it's impossible to design a universal algorithm that can determine if any given TM is a Busy Beaver machine.

To put things into scale, let's investigate how the properties of the class of Busy Beavers are different from the class of Ramsey Numbers, which also grow quite fast.

Theorem 5.3. *The Ramsey Numbers of the form $R(k, k)$ are computable.*

Proof. We prove this theorem by showing that the problem Clique, defined as

$$\textsc{Clique} = \{\langle G, k \rangle\} : \text{ graph } G \text{ contains a monochromatic clique on } k \text{ vertices},$$

is decidable.

We construct a Turing machine to check if a pair $\langle G, k \rangle$ is in Clique. This Turing machine would simply consider all possible combinations of k nodes and consider the edges that connect those nodes. If they form a monochromatic clique, then accept. If not, then move on to the next combination of k vertices. If no combination of k vertices forms a clique, then reject.

Note that since there are at most $\binom{n}{k} = O(n^k)$ combinations to consider, this Turing machine runs in polynomial time. Thus, because it is possible to verify that a pair $\langle G, k \rangle \in$ Clique, Clique $\in \mathsf{NP}$, and is therefore decidable.

n	$R(n,n)$	$S(n)$
1	1	1
2	2	6
3	6	21
4	18	107
5	$43 \leq R(5,5) \leq 49$	$S(5) \geq 4.7 \cdot 10^7$
6	$102 \leq R(6,6) \leq 165$	$\mathrm{S}(6) \geq 8.6 \cdot 10^{15}$

FIGURE 2. A tabular representation for easy comparison between the values of the Ramsey Numbers versus the Busy Beaver Numbers.

Hence, the Ramsey Numbers are computable.

□

But what does this really mean? To show the drastic difference between a computable and non-computable function, we have organized the values in Table 2. It is impossible to design a TM that can compute the value of $S(n)$ without running all possible n-state TMs and comparing their runtimes. What this means is that The Busy Beaver Problem is not even in the class NEXP, the class of problems that can be solved by an NFA in exponential time. In fact, not even quantum computing classes would encompass the Busy Beaver problem. What this implies is that the only way to calculate $S(n)$ for some N is by guessing-and-checking through all $(4n+1)^{2n}$ possible TMs and evaluating the maximum runtime. However, because such TMs tend to run for extended periods of time, values of $S(5)$ and above have not been confirmed yet. And, as we saw in the previous section, some values we will never be able to confirm (at least under the commonly used axiom systems).

Remark 2. Even though this section mentions the "cool" uncomputability of the Busy Beaver problem, this is what makes the problem so hard.

$m\backslash n$	0	1	2	3	4
0	1	2	3	4	5
1	2	3	4	5	6
2	3	5	7	9	11
3	5	13	29	61	125
4	13	65533	$2^{65536}-3$	$2^{2^{65536}}-3$	$2^{2^{2^{65536}}}-3$

FIGURE 3. Some values of the Ackermann-Péter function for small m, n. Values from [9].

6. The Ackermann Function

In the spirit of discussing fast-growing functions, we proceed to discuss another bohemoth: the Ackermann function.

The name "Ackermann function" is a term given to a large family of functions that are all variants of the same idea. Yet, any recursive function along the lines of the Ackermann function would serve as one of the fastest-growing computable functions. One variant, known as the Ackermann-Péter (AP) function, is defined as follows:

Definition 6. Let $A(m, n)$ denote the value of the Ackermann-Péter function for $m, n \in \mathbb{N}$. Then,

$$A(m,n) = \begin{cases} n+1 & m{=}0 \\ A(m-1,1) & m>0 \text{ and } n=0 \\ A(m-1,A(m,n-1)) & m,n>0. \end{cases}$$

Some values of $A(m, n)$ for small m, n are included in Figure 3.

Clearly, it is possible to compute values of the AP function: we simply define a TM that runs the AP recursion on any given input (m, n). However, the AP function is a special kind of recursive computable function, in the sense that it is not primitive recursive (meaning one cannot code the AP function using a series of for loops, such as the Fibonacci recursion).

Thus, if we were to pit the AP function up against the Busy Beaver function, the latter would clearly outrace the AP function in terms of growth, as it is uncomputable. However, one remarkable thing to note is that the only arithmetic operation in the AP function is the addition of 1. Thus, its immensely quick growth comes simply from the addition of a bunch of 1s. This can be explained, however, by the fact that the runtime of the AP function must be at least as long as the *value* of the input (not its length) [10].

6.1. The Inverse Ackermann Function.

Even though the AP function itself grows immensely quickly, its inverse function grows very slowly, in addition to having some other interesting properties and uses.

Before defining the inverse Ackermann function, however, we must define the following function:

Definition 7. Let α denote a class of functions, so that for each $i \in \mathbb{N}$, α_i is a function in this class. We define these α_i functions in the following manner:

- $\alpha_1(n) = \lceil n \rceil$ for n ≥ 1

- $\alpha_k(1) = 0$ for $k \geq 2$
- $\alpha_k(n) = 1 + \alpha_k(\alpha_{k-1}(n))$ for $k, n \geq 2$

It can be shown that $\alpha_i(n) \geq \alpha_{i+1}(n)$ for all n. Furthermore, after testing some small values, it is not too difficult to see that eventually, regardless of the value of n, there exists a k such that for all $j \geq k$, sequence $\alpha_j(n) = 3$. The single-input inverse Ackermann function is interested in such a value of k and is defined in the following manner:

Definition 8. Define the inverse Ackermann function $\alpha(n)$ to be

$$\alpha(n) = \min_{\alpha_k(n) \leq 3} k$$

for any positive integer n.

However, the 2-input inverse Ackermann function is defined in a slightly different (though similar) manner [12]:

Definition 9. $\alpha(m, n) = \min_{\alpha_k(n) \leq 3 + \frac{m}{n}} k$.

I would like to point out the following property, though a proof of it is outside of the scope of this paper (further reading at [11]):

Proposition 1. *The inverse Ackermann function is computable in linear time.*

This property becomes handy, especially in algorithms such as the union-find path compression, which runs in in inverse Ackermann time. Recall that the union-find path compression algorithm involves linking two sets of nodes by their roots, while in the process contracting each of the sets so that each node within each set points to the node of its respective set.

Say we have a total of n vertices and m queries. Then, it turns out that the time complexity of a union find path algorithm that runs on this input has time complexity $O((m + n)\alpha(n))$[1].

[1]A summarized proof of this is provided by -is-this-fft- on CodeForces at https://codeforces.com/blog/entry/98275. However, a more detailed and technical proof can be found in Tarjan's article titled *Efficiency of a Good But Not Linear Set Union Algorithm.*

7. Acknowledgements

Thanks to Simon Rubinstein-Salzedo and the Euler Circle institution for providing me with a platform to investigate higher-level mathematics. His valuable guidance and suggestions were crucial during the process of making this paper. I would also like to thank Sherry Sarkar for her insightful feedback and suggestions that helped shape this paper.

References

[1] Aaronson, S. (2020). *The busy beaver frontier.* ACM SIGACT News, 51(3), 32-54.

[2] Pavlus, J. (2020, December 22). *How the slowest computer programs illuminate math's fundamental limits.* Quanta Magazine. https://www.quantamagazine.org/the-busy-beaver-game-illuminates-the-fundamental-limits-of-math-20201210/

[3] Harland, J. (2005, August 16). *The Busy Beaver, the Placid Platypus and other Crazy Creatures.* Busy beaver. Retrieved December 12, 2022, from http://titan.csit.rmit.edu.au/ e24991/busybeaver/seminarAug05.html

[4] Cover, In Gopinath, Bhaskarpillai Chaitin, Gregory. (2003). *Computing the Busy Beaver Function.*

[5] O'Rear, Stefan. Construction for the 748-state Turing machine that halts if and only if ZF set theory is inconsistent. https://github.com/sorear/metamath-turing-machines/blob/master/zf2.nql

[6] Matiyasevich, Yuri O'Rear, Stefan Aaronson, Scott . Construction for the 744-state Turing machine that halts if and only if the Reimann Hypothesis is false. https://github.com/sorear/metamath-turing-machines/blob/master/riemann-matiyasevich-aaronson.nql

[7] Anonymous. Construction for the 27-state Turing machine that halts if and only if the Goldbach Conjecture is false. https://gist.github.com/anonymous/a64213f391339236c2fe31f8749a0df6

[8] Michel, P. *Busy beaver competition and Collatz-like problems.* Arch Math Logic 32, 351–367 (1993). https://doi.org/10.1007/BF01409968

[9] *Ackermann function - Saylor Academy.* (n.d.). Retrieved December 12, 2022, from https://resources.saylor.org/wwwresources/archived/site/wp-content/uploads/2011/06/Ackermann-Function.pdf

[10] *Ackermann function - definition and properties.* Ackermann Function - Definition and Properties — Definition Properties. (n.d.). Retrieved December 12, 2022, from https://www.liquisearch.com/ackermann_function/definition_and_properties#:~:text=One%20interesting%20aspect%20of%20the%20Ackermann%20function%20is,its%20output%2C%20and%20so%20is%20also%20extremely%20huge.

[11] Sureson, C. (2021). *The inverse of Ackermann function is computable in linear time.* Fundamenta Informaticae, 182.

[12] Nivasch , G. (n.d.). *Gabriel Nivasch - Inverse Ackermann.* Retrieved December 12, 2022, from https://www.gabrielnivasch.org/fun/inverse-ackermann

Euler Circle, Palo Alto, CA 94306
Email address: stemanant@gmail.com