the association for computational heresy

presents

a record of the proceedings of

SIGBOVIK 2022

the sixteenth annual intercalary robot dance party in celebration
of workshop on symposium about 2^6th birthdays; in particular,
that of harry q. bovik

cover art by someone probably

carnegie mellon university

pittsburgh, pa

april 8, 2022

ach

Association for
Computational Heresy

Advancing computing as Tomfoolery & Distraction

SIGBOVIK

A Record of the Proceedings of SIGBOVIK 2022

ISSN 2155-0166

April 8, 2022

Additional copies of this work may be ordered from Lulu; refer to `http://sigbovik.org` for details.

SIGBOVIK 2022

Message from the Organizing Committee

```
\Message2020{16}{uenchiest}{$2^6$}
```

TODO: Tell next year's committee that they should probably avoid further milking the blindsight joke

: Programming Languages **5**

1 Modernized Python 6

2 Grand Challenges in Programming Languages Position Paper: What, if anything, does multiplication even mean? 13

3 A Type-and-Affect System for Semisignificant Whitespace 16

4 brainfuck++: a much needed extension to brainfuck 23

: Other Languages **33**

5 Tironiculum: Latin Speech Recognition via Latin Text-to-Speech 34

6 Abecedarial Acrostic, Alphabetized Amusingly Because Beings Blissfully Cause Celebratory Centennials... (Note: Full Title is Longer) 40

7 On "Ra-men, Ra-men ramen ramen" 41

8 ACTION: A Catchy Title Is all yOu Need! 42

9 A Deep Learning Approach for Deeply Inaccurate Wordle Solving 43

: Systems **45**

10 Edward, edMUnD & Edwin: Line-Based Text Editing for the 21st Century 46

11 A Free Computer Vision Lesson for Car Manufacturers or It is Time to Retire the Erlkönig 52

12 Redundant Coupling 54

: Theory **57**

13 Destructive Logic 58

14 Overlap-Maximal Graph Labelings: Graph Labelings with Non-Disjoint Vertex and Edge Sets (and how they can be used for encryption, poetry, and breaking mathematics) 62

15 Neo-classical Logic: Une Logique non classique 69

16 A Patriotic Analysis of Programming Paradigms 76

17 On Ruinment: Ruination Theory and its Consequents 84

: Astrophysics **91**

18 Method and Tool for Estimating the Mass of the Black Hole Located in the Office of Immigration, Refugees and Citizenship Canada Causing a Supermassive Time Dilation in the Visa Extension Process 92

19 Black Hole Computation 102

20 Schrödinger's SAT: Generalizing Quantum Bogosort to Prove P = NP Under Many-Worlds Quantum Mechanics 109

21 Solving Double Execution of Java's paint() Method by Counting Down to the Heat Death of the Universe (plus language compendium) 111

: **Climate Science** **139**

22 Towards Cloud Computing 140
23 Ecological Memory Management: Beyond Garbage Collection 144
24 Infrastructure-as-PowerPoint: A No-Code Approach to Cloud Infrastructure Management 166
25 On the Possibilities and Challenges of Organic UAV-Assisted MEC 172

: **A Brief Musical Interlude** **177**

26 Baby Sharks are More than Sharks . They are Earworms Sung More than Happy Birthday Perhaps 178
27 Everybody Clap Your Hands: The Cha Cha Slide is Turing Complete 184
28 Exhaustive Survey of Rickrolling in Academic Literature 189

: **Best Practices** **201**

29 Quadruple-Blind Peer Review 202
30 Optimal degeneracy through OwO based variable names 206
31 Multiplication by repeated addition, with fraction handling. 208
32 Objective Correlation Metrics for Quality of Code Estimation 209

: **Algorithmic Advances** **215**

33 Functorial wrappers for high-dimensional classification algorithms 216
34 (Un)helper functions 220
35 A sometimes-accurate O(1) search algorithm 223

: **Mathematic Retreats** **225**

36 Improved Data and Instruction Locality in Long Division 226
37 The New New Math: using sentiment analysis of mathematics word problems to gauge children's reactions to teaching 8-bit floating point arithmetic for the new California public school math curriculum 229
38 Infix Modifiers for Flexible Multiplication 236

: **Aesthetics** **239**

39 Real-Time Foliage Simulation 240
40 Using deep CNNs to prove that I look better than Tom Cruise and Shah Rukh Khan combined 241
41 EBMP: Efficient Bitmap Encodings on Ethereum Virtual Machines 243
42 A Machine Learning Approach To Classifying Cuteness 251
43 Attractiveness Learning: A General Solution for the Cold-Start Problem . . 253

: **Hardware** **257**

44 Correct-It-Yourself Paper Updater 258
45 Harder Drive: Hard drives we didn't want or need 259
46 Just a Regular Paper 278
47 Wearable RF-shield repositories 279

: Black Boxes 281

48 A 23 MW data centre is all you need . . . 282
49 When Pull Comes To Shove... Do Both! . . . 293
50 man exorcism . . . 338
51 2x a fake submission . . . 340

: SIGBOVIK 343

52 Analysis of A New Error Calculation Technique . . . 344
53 A Third Thorough Investigation of the Degree to which the COVID-19 Pandemic has Enabled Subpar-Quality Papers to Make it into SIGBOVIK, by Reducing the Supply of Authors Willing to Invest the Necessary Effort to Produce High-Quality Papers . . . 345
54 this is not a review . . . 346

Programming Languages

1 Modernized Python

Daniel Ng

Keywords:programming languages, type theory, python, sml

2 Grand Challenges in Programming Languages Position Paper: What, if anything, does multiplication even mean?

Jim McCann

multiplication, punctuation: *, punctuation: x, punctuation: ., grand challenges

3 A Type-and-Affect System for Semisignificant Whitespace

Stefan Muller

Keywords: whitespace, white space, white nspace

4 brainfuck++: a much needed extension to brainfuck

Cédric Ho Thanh

Keywords: Programing langage, brainfuck, concurrency, machine learning

Modernized Python

Daniel Ng

March 25, 2022

Abstract

The popular Python programming language is generally heralded for its usability and learnability. However, this often comes at the cost of efficiency. Solutions to this problem have emerged over the last few years to impose static type systems onto Python without intruding on the underlying syntax. This paper deviates from the norm of building on the existing language, and instead looks to completely overhaul the concrete syntax while maintaining its beloved semantics.

Introduction

Python has recently surged in popularity, largely for two reasons. Firstly, it is an easy language for beginners to learn, especially because its array of modules allows for relatively complex projects to be built up relatively quickly. Additionally, these modules are often used in fields such as artificial intelligence and machine learning as a reliable and easy-to-use language. However, concerns from type theorists have emerged, largely regarding its inefficiency and unsafety.

The approach of overlaying a type system onto the existing concrete syntax is one that has been considered in several places previously. Type theorists claim that this could speed up the language and allow for fewer runtime errors. However, this approach has a negative impact in certain edge cases. Certain scripts may wait for Python programs to terminate, causing processor cycles to be wasted. Additionally, speeding up Python gives programmers less time to slack off while their code is running, the social drawbacks of which are immense.

It is thus only natural to look into ways to make the Python programming language more palatable for functional programmers. Current approaches focus on keeping Python's concrete syntax while overhauling the abstract syntax to better match that of functional languages. In contrast, we try a revolutionary new approach in this paper and try to improve the concrete syntax make it more palatable for functional programmers. In the spirit of Harper [1], we therefore propose a Modernized Python, which seeks to eliminate the issues with Python at the level of concrete syntax, while maintaining both the speed and memory usage of original Python. Any resemblance to real languages, living or dead, is purely coincidental[1].

Problems

Concrete Syntax

While the concrete syntax of Python seems both easy to learn and easy to write, its actual nature is really not like that at all. Consider the behaviour of the following program:

```
def fact(x):
   if (x == 0):
      return 1
  return x * fact(x - 1)
print(fact(10))
```

At first glance, this appears to be a perfectly legal[2] program that computes the factorial of a given positive number. However, upon actually attempting to interpret this code, it returns `IndentError` rather than printing `3628800`. Some might argue that this has to do with the use of a three-space indent for some awful reason. However, the same problem exists for users of the much more popular two-space indent. It follows from these strict indentation requirements that handwritten Python code on paper[3] is extremely unlikely to run, much to the dismay of many exam-takers.

The additional requirement that all lines be terminated with a newline adds further complications to the concrete syntax, especially when trying to write long commands and avoid confusion. The indentation issue is also exascerbated when determining where a loop ends or begins. Marking the end of code blocks in general is therefore another concern that we will attempt to address when modernizing Python.

One feature that must be preserved through this is Python's acceptance of generally questionable code. For example, the following function would be rejected by most sensible type systems, yet is allowed by Python:

```
def f(_):
    return f(f)
```

Python will attempt to execute `f(3)` in vain, eventually reaching a `RecursionError`, generating thousands of call stack frames in the process before Python realizes what is going on

[1] I promise.

[2] Except in the state of California, where this code is known to cause cancer, birth defects, or other reproductive harm. And only in the state of California.

[3] Or on whiteboards, cuneiform tablets, The Fence, or any other non-digital media.

and averts the impending memory disaster.[4] Python's willingness to try once again leads to run-time errors, but it gets an A$^+$ for effort. This characteristic tenacity is preserved even when fixing the above problems, at the cost of efficiency.

Statics and Dynamics

We now look to the statics and dynamics of Python to better understand how it behaves.

Statics:

$$\frac{}{\Gamma \vdash x : \texttt{PyObject}}$$

Everything is a `PyObject`, so this is just stating the obvious. There are no further statics, because no other types exist anyway.

Dynamics:

$$\frac{}{x \text{ final}} \qquad \frac{}{\texttt{TypeError} \text{ final}} \qquad \frac{}{f(x) \longmapsto \texttt{TypeError}}\text{(With high probability)}$$

The first dynamic just states that if we have an established `PyObject`, then there is nothing further to be done to it, so we just leave it alone. Likewise, if we have managed to reach a `TypeError`, there is nowhere further we can go, as the program has reached an erroneous state.

The third rule states that most expressions will result in a `TypeError`. This is evidenced by the fact that many `PyObject`s are functions, however, the probability that the argument x has the correct type to be input to f without causing a `TypeError` is rather low, since the number of total `PyObject`s is quite large and the set of valid inputs S is likely significantly smaller[5]. Rudimentary calculations show that the probability of a `TypeError` can exceed 99%, which sounds like quite a high probability.

We now carry out the time-honoured tradition of proving progress and preservation.

Progress: If $\Gamma \vdash x : \tau$, then either x final or $x \longmapsto x'$ for some x'. For this, we need a canonical forms lemma[6]. Fortunately, we can note that every `PyObject`consists of either no function applications or at least one function application, so if x : `PyObject` then either $x = y$ for some non-function application y or $x = f(z)$ for appropriate f and z. The first case uses the first dynamic to conclude that x final. The third rule indicates that $f(z) \longmapsto$ `TypeError`. ■

[4] For dealing with this, the author recommends downloadmoreram.com.

[5] If there are $|S| = k$ valid inputs, there are also at least k^k invalid ones, all functions mapping S to itself.

[6] Or Canadian Football League.

Preservation: If $\Gamma \vdash x : \tau$ and $x \longmapsto x'$, then $\Gamma \vdash x' : \tau$. Consider that $\tau = \texttt{PyObject}$ since `PyObject` is the only possible type. Additionally note that the only possible rule that could produce the judgement is the third dynamic, which implies that $x' = \texttt{TypeError}$. Finally, note $\Gamma \vdash \texttt{TypeError} : \texttt{PyObject}$ by the statics. ■

The rules for Python are therefore not particularly complex. This is mostly evidenced by the fact that the lines in the middle of the rules are not particularly long – more detailed languages tend to use rules with lines spanning nearly the width of the page. The restrictions that these simplistic rules create are, according to Harper [1], what makes Python such a powerful language[7].

Basic Changes

We begin this section with a snippet of Python code which takes the digital root of an integer.

```
"""
Find the digital root of the number.
"""
def dr(x):
    res = 0
    while(x > 0):
        res += x % 10
        x = x // 10
    if(res >= 10):
        return dr(res)
    return res
```

We begin with the most important part of the code: the ~~arithmetic~~ ~~control flow~~ comments. Nesting comments can be somewhat cumbersome, as trying to comment out the entire function can lead to uncommenting parts of the code. Comments therefore now start and end with `(* *)` to create clear start and finish markers for a comment.

To make it apparent that `dr` is a function and `res` is a variable, we can rewrite them with keywords that indicate what they are, while still not forcing them to take on certain types. We also use braces in place of indentations to keep code blocks organized as the program evolves. Finally, we make arithmetic more readable by using words to describe what is going on.

```
(* Find the digital root of the number. *)
fun dr(x) {
  val res = 0
```

[7] In terms of both computational power and energy usage – the technique of using Python scripts as space-heater extensions for laptops has become increasingly prevalent.

```
  while(x > 0) {
    res += x mod 10
    x = x div 10
  }
  if (res >= 10) return dr(res)
  return res
}
```

At this ‘point’, the code is not exactly sure what it wants to look like. Some of the Python has been stripped away from the concrete syntax while maintaining the same underlying interpreter. We therefore introduce the following loop construct for a `while` loop. Notice how this looks functional since the “mutable state” is hidden away in the function arguments.

```
fun while (x, state) guard body = if guard x
                                  then while (body (x, state))
                                     guard body
                                  else state
```

Additionally, the braces have been stripped away for clarity reasons, now that our code is organized again. The `if/then/else` and `let/in/end` constructs replace them when needed to keep code organized.

```
(* Find the digital root of the number. *)
fun dr(x) = let
  fun body (a, b) = (a div 10, b + (a mod 10))
  val res = while (x, 0) (fn x => x > 0) body
in
  if res >= 10 then dr(res) else res
end
```

Notice the arrow `=>`, which looks like an elongated musical accent and is therefore used to add emphasis to the loop guard. Eliminating unnecessary whitespace allows us to compress our code further, attaining the goal of the Single Massive Line (SML)[8].

```
fun dr x=let fun body(a,b)=(a div 10, b+(a mod 10))val res=
 while(x,0)(fn x=>x>0)body in if res>=10 then dr res else res
  end
```

Further Changes

Here are some further constructs that can be added to the language to allow for easier compilation from SML-like constructs to Python, in order to maintain the same efficiency associated with Python compilation.

[8] Whitespace-equivalence is not a well-studied topic, likely due to people attempting to read code.

```
   filter p L ==> [x for x in L if p(x)]
      map f L ==> [f(x) for x in L]
foldl f acc L ==> x = acc
                  for y in L:
                      x = f(acc, x)
                  return x
```

The widespread importing of Python modules is closely related to SML's module system, though some might argue it is in name only. A syntax similar to that of the SML module system is therefore used in Modernized Python. However, the conversion of many type-checking failures to run-time errors eliminates the need for signatures, which are replaced by ~~the user's choice between reading documentation and~~ trial-and-error.

This error conversion also enables the user to write programs such as:

```
fun f () = if random() > 0.5 then 15 else "potato"
fun g _ = 3 + (f ())
```

The user is therefore able to gamble the stability of a currently running program by attempting to use the result of a call to `f`, for no apparent reward at all. Use cases for this seem extremely limited and further research is necessary. At the very least, allowing this sort of 'compilation' to Python allows well-typed portions of programs to be tested before the remainder of the program is written in full.

Pedagogy

The following introductory computer science courses at CMU were surveyed about their opinions on Modernized Python.

- 15-122: Honk honk honk honk honk? (Translation: Does it include contracts?)
- 15-150: My code speed has gotten slower, but maybe converting it to CPS will somehow help.
- 15-210: Modernized Python could go pretty quickly on multiple processors with the right libraries if we put it on Diderot.
- 15-213[9]: Do not SIGBOVIK 213.
- 15-251: Not what we meant when we said to perform a reduction.

15-112 did not need to be surveyed, since they already used Python before its modernization.

[9] https://www.google.com/search?q=15213

Conclusion

By restructuring the concrete syntax, we have therefore managed to translate code written in a functional style to Python. In doing so, users are now able to run a subset of SML in a much slower manner. This thrilling discovery allows us to combine the dynamic type-checking and general permissibility of Python with SML's tidy syntax. By taking the opposite approach to the popular literature, we are therefore able to develop a revolutionary new system which makes the Python language Slightly More Likeable.

References

[1] Robert Harper. 2016. Practical Foundations for Programming Languages (2nd. ed.). Cambridge University Press, USA.

Grand Challenges in Programming Languages Position Paper:
What, if anything, does multiplication even mean?

Jim McCann
Programming Studies Institute
Cranberry Lemon University
Pittsburgh, PA 15213
ix@tchow.com

Abstract

What, if anything, does multiplication even mean? Current programming languages answer the question in quirky and overly specific ways, which – like being forced to wear socks only on your feet – seems fine until you encounter a situation where it does not. In this position paper, I will describe the ambiguity implicit in "multiplication," how the scope of defining a proper multiplication is so broad that it is inconceivable that within any ten lifetimes of good work it could be solved, and conclude that – therefore – defining a sufficiently flexible multiplication operator is pretty much impossible.

1 Introduction

What, if anything, does multiplication even mean? While the definition of addition is abundantly clear, like Crystal Pepsi; the proper definition of multiplication is terribly muddy, like Pepsi. Multiplication has so many possible meanings (Section the next section) that I cannot begin to imagine a world where programming languages have a uniform, sensible, interpretation of the operation, just like I can't begin to imagine golf negative trait § couch the.

Therefore, I claim for the throne of the SIGBOVIK Grand Challenges in Programming Languages Position Paper Grand Challenges List (Appendix A) the challenge of defining and implementing a consistent and flexible multiplication semantics in any modern programming language.

the next section Background

What, if anything, does multiplication even mean? Nobody really knows, but a lot of people have tried to do something smart anyway and got it sort of wrong but not wrong enough to really matter, just like you did when you pretended to understand how to use the salad bar at a Ponderosa steakhouse.

Just a few options:

1.1 Addition in the Log Domain

Slide rule users may attempt to use the so-called identity:

$$a * b := \exp(\log(a) + \log(b)) \tag{1}$$

But any savvy numerical methodist will realize that the exp and log functions are simply shorthand for a near-infinite amount of multiplication:

$$\exp(x) := 1 + \frac{x}{1} * \left(1 + \frac{x}{2} * \left(1 + \frac{x}{3} * (1 + \ldots)\right)\right) \tag{2}$$

Meaning that the "definition" in (1) is clearly circular.

-1st Annual Symposium on Harry Q. Bovik (SIGBOVIK 2022), Pittsburgh, PA.

1.2 Replication

Some languages decide that multiplying a number by a string should replicate the string:

```
>>> 5 * ’x’
’xxxxx’
```

But this definition is inconsistent, since numbers aren’t properly promoted to strings:

```
>>> 5 * 5
25
#expected: ’55555’
```

and, further, the operator is broken for fractions:

```
>>> 5.5 * ’x’
TypeError: can’t multiply sequence by non-int of type ’float’
#expected: ’xxxxx›’
```

1.3 Linear Algebra

Working in a vector space $\mathbb{V}$ over field $\mathbb{F}$ exposes one to a wide variety of multiplications.

By definition, every vector space allows scalar-vector multiplication $* : \mathbb{F} \times \mathbb{V} \to \mathbb{V}$. For example, in $\mathbb{R}^3$,

$$a * (b_1, b_2, b_3) := (a * b_1, a * b_2, a * b_3) \tag{3}$$

But many vector spaces also have an inner product $* : \mathbb{V} \times \mathbb{V} \to \mathbb{F}$. In $\mathbb{R}^3$,

$$(x, y, z) * (s, t, r) := x * s + y * t + z * r \tag{4}$$

And some vector spaces – like $\mathbb{R}^3$ have a cross product $* : \mathbb{V} \times \mathbb{V} \to \mathbb{V}$,

$$\begin{aligned}(a_1, a_2, a_3) * (b_1, b_2, b_3) := (\quad & a_2 * b_3 - a_3 * b_2, \\ & a_3 * b_1 - a_1 * b_3, \\ & a_1 * b_2 - a_2 * b_1 \quad)\end{aligned} \tag{5}$$

Not to mention the element-wise product $* : \mathbb{V} \times \mathbb{V} \to \mathbb{V}$,

$$(a_1, a_2, a_3) * (b_1, b_2, b_3) := (a_1 * b_1, a_2 * b_2, a_3 * b_3) \tag{6}$$

And vector spaces may have an associated space of linear transformations, which in finite-dimensional vector spaces may be notated using matrices $A \in \mathbb{F}^{m \times n}$ that tabulate the action of the transformation in some basis:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \tag{7}$$

Meaning that linear transformations $A \in \mathbb{F}^{r \times m}$ and $B \in \mathbb{F}^{m \times c}$ can be composed by multiplication $* : \mathbb{F}^{r \times m} \times \mathbb{F}^{m \times c} \to \mathbb{F}^{r \times c}$:

$$\begin{gathered} A * B := C \\ \text{where } c_{ij} := \sum_{k=1}^{m} a_{ik} * b_{kj} \end{gathered} \tag{8}$$

And, further, these linear transformations can be *applied* to vectors with multiplication $*$: LaTeX Gold Warning: MathBBFree per-document usage limit exceeded; upgrade to remove this limitation. $F^{m \times n} \times F^n \to F^m$:

$$\begin{gathered} A * b := c \\ \text{where } c_i = \sum_{k=1}^{n} b_i A_{ik} \end{gathered} \tag{9}$$

Further, it is often useful to be able to compose scalar multiplication with linear transformations using multiplication $* : F \times F^{r \times c} \rightarrow F^{r \times c}$:

$$\begin{gathered} a * B := C \\ \text{where } c_{ij} = a * b_{ij} \end{gathered} \tag{10}$$

Which is seven different definitions of $*$, all of which depend on each-other as well as some unstated $*$'s that came along with the definition of F and the vector field.

1.4 Unary Multiplication

Of course, there is no rule that $*$ must be a binary operator. Some programming languages, like C++, provide unary $*$ operators which – like unary $+$ – appear to do nothing:

```
int x();
assert(+x == x); //unary addition is identity
assert(*x == x); //unary multiplication is identity
//assert(++x == x); //compile error?!
assert(**x == x); //still the identity
```

2 Potential Approaches

What, if anything, does multiplication even mean? My editor suggests that I should provide some approaches to this seemingly insurmountable problem but, honestly, I just can't think of any possible ideas. I suppose we could stick with the *status quo* – multiplication meaning something which is right in some circumstances and not right in others – but I can't see this *quo*, well, *status*-ing.

I think the revolution is coming, like a rising tide, and I'm both apprehensive and elated by the prospect of what may lie on the other side, like the things the rising tide casts upon the beach; but I sure as heck don't want to step in some of the things. So I'll be walking carefully, and so should you.

3 Conclusions

What, if anything, does multiplication even mean? We may never know, and perhaps it's for the best that we do not. Like when your parents come home drunk and confess their love to your dog and you realize as it's happening that you don't have a dog and it's you who are drunk. And you just said it all out loud.

Current programming languages seem to want to make the decision for you, but they certainly don't know best. We are all grown up and they can't make us go to bed early.

Acknowledgments and Disclosure of Funding

This paper was produced with the help of a fictitious bribe from several shady treeacters, which are like characters, but also trees. Ent it just the way!

A The SIGBOVIK Grand Challenges in Programming Languages Position Paper Grand Challenges List

1. Defining a sensible notion of multiplication in any modern programming language.
2. Et cetera.

A Type-and-Affect System for Semisignificant Whitespace

Stefan Muller
Illinois Institute of Technology

```
if (p != NULL) {
  printf("%d\n", *p)
}
```

```
if (p != NULL)
{
  printf("%d\n", *p)
}
```

Figure 1: If you're a C programmer, one of these looks beautiful and one of them makes you want to vomit.

```
if (p != None):
  print("p␣is␣" + p + "␣and␣oh␣no␣I'm␣getting␣close␣to␣the␣end␣of␣the␣line␣can␣I␣put␣a␣linebreak␣here?␣I␣don't␣know")
```

Figure 2: Python syntax terrifies me.

ABSTRACT

Whitespace characters are generally ignored by compilers for most languages. But for something ignored by the compiler, programmers sure do spend a lot of time arguing about the use of whitespace in programs. In this paper, we show whitespace some (tough?) love and enforce some of your favorite pedantic style conventions statically. Oh, you asked if this is like a linter or something? Not exactly...

1 INTRODUCTION

Most languages (with the notable exception of Whitespace [3]) treat whitespace characters as insignificant other than as delimiters between tokens, and discard them during lexing. That's right, whitespace characters don't even make it to the parser, that's how overlooked they are. And yet, whitespace is related to so many of the features that many programmers and programming instructors spend so much time lecturing and/or arguing about: every programming course, company, book author, and pedant in general has a style guide for their preferred language. Many of these style guides give advice on line length, indentation, when to start a new line before and after delimiters, and other features directly or indirectly related to whitespace (newlines, spaces, and, if you're a terrible person, tabs), as seen in Figure 1. Some provide good advice on creating reasonable code, others harp on pet peeves, and many are contradictory (e.g., [2, 4]). Few languages make these guidelines standard or enforceable, and some languages even make good style difficult. It is an irony of programming languages that languages which do not have "significant whitespace" have no way of enforcing good style. Meanwhile, languages in which whitespace characters are signficant can enforce some stylistic conventions (e.g., indentation in Python), but the very fact that whitespace is significant makes it difficult to achieve other elements of good style (who knows when you can break a long line in a Python program without wreaking havoc?, Figure 2).

$$
\begin{array}{lcl}
\tau & ::= & \text{int} \mid \text{bool} \mid \tau \times \tau \mid \tau \to \tau \\
\underline{v} & ::= & \top \mid \bot \mid \mathrm{I} \mid \dagger \mid \emptyset \\
e & ::= & \overline{n} \mid \text{true} \mid \text{false} \mid \text{fun}\text{␣}x\text{␣}{-}{>}\ e \mid e\text{␣}e \mid (e, e) \\
 & & \mid \text{let}\ (x, \text{␣}y) = e\ \text{in}\ e \mid \text{let}\text{␣}x\text{␣} = e\text{␣}\text{in}\ e \mid \text{if}\ e\text{␣}\text{then}\ e\text{␣}\text{else}\ e \\
 & & \mid \text{␣}e \mid e\text{␣} \mid \backslash\text{n}e \mid e\backslash\text{n}
\end{array}
$$

Figure 3: The syntax of Pedant

In this paper, we begin by presenting a type system Pedant that enforces good coding style with regard to whitespace, drawing on the best of the significant and insignificant whitespace worlds. In a normal paper introduction, we would now go on to describe how we do this and the results we achieve, but as a result, most academic papers are very uniform as to narrative style. It would be like if every novel took the approach of beginning the first chapter with the protagonists reflecting back on the events that are to be described in the novel. This can be a good storytelling technique, but it doesn't work well if overused. Instead, I'll start with a more traditional narrative style that builds a higher degree of suspense. And so we set off on this adventure together.

2 PEDANT: A SYNTAX AND TYPE SYSTEM FOR ENFORCING PEDANTIC STYLE RULES

In this section, we begin to develop Pedant, a language equipped with a type system to enforce the pedantic style rules we discussed in the introduction. Figure 3 gives the syntax of Pedant, an ML-style calculus extended with visible whitespace (␣, \n). The calculus consists of integers, Booleans, lambdas, pairs, and the relevant elimination forms (including let-binding for pairs).

The typing judgment for Pedant is $\Gamma \vdash e : {}_{m}\tau^{\underline{v}}_{n}$, meaning that under variable context Γ, the expression e has type τ, starts with m columns of whitespace, and is at most n columns wide. The type also includes a description $\underline{v}$ of e's newlines, which can be $\top$ ("top", starts with a newline), $\bot$ ("bottom", ends with a newline), I ("both", starts and ends with a newline), $\dagger$ ("internal", does not start or end with a newline but is not all on one line), and $\emptyset$ ("none", is all on one line).

Figure 4 gives the typing rules for Pedant. The variable rule T-Var looks x up in the context and types x with the given type. Its width is $|x|$, the number of characters in the variable name. We assume variable names do not contain whitespace, so the left is 0 and the newline description is $\emptyset$. The introduction rules for integers and booleans are straightforward as well. The remaining rules are standard as far as the introduction and elimination of types τ goes, but the whitespace-relevant parts of the rules are somewhat[1] nonstandard, and so we'll describe them very briefly before going way too quickly on to the next section of the paper. As an example, there are three rules for "if". Rule bool-E1 assumes that

[1] I'm hoping there will be an award for "biggest understatement" this year

SIGBOVIK 2022,
2022.

$$\frac{}{\Gamma, x:\tau \vdash x : {}_{0}\tau^{\emptyset}_{|x|}}\ (\textsc{Var}) \qquad \frac{}{\Gamma \vdash \overline{n} : {}_{0}\mathsf{int}^{\emptyset}_{|\overline{n}|}}\ (\text{int-I}) \qquad \frac{}{\Gamma \vdash \mathsf{true} : {}_{0}\mathsf{bool}^{\emptyset}_{4}}\ (\text{bool-I1}) \qquad \frac{}{\Gamma \vdash \mathsf{false} : {}_{0}\mathsf{bool}^{\emptyset}_{5}}\ (\text{bool-I2})$$

$$\frac{\Gamma, x:\tau_1 \vdash e : {}_{m+1}\tau_2{}^{\emptyset}_{n}}{\Gamma \vdash \mathsf{fun}\textvisiblespace x\textvisiblespace -> e : {}_{0}\tau_1 \longrightarrow \tau_2{}^{\emptyset}_{4+|x|+4+m+n}}\ (\to\text{-I1}) \qquad \frac{\Gamma, x:\tau_1 \vdash e : {}_{m+1}\tau_2{}^{\top}_{n}}{\Gamma \vdash \mathsf{fun}\textvisiblespace x\textvisiblespace -> e : {}_{0}\tau_1 \longrightarrow \tau_2{}^{\dagger}_{\max(m+1+n,4+|x|+4)}}\ (\to\text{-I2})$$

$$\frac{\Gamma \vdash e_1 : {}_{m}\tau_1 \longrightarrow \tau_2{}^{\emptyset}_{n} \qquad \Gamma \vdash e_2 : {}_{j}\tau_1{}^{\emptyset}_{k}}{\Gamma \vdash e_1\textvisiblespace e_2 : {}_{m}\tau_2{}^{\emptyset}_{n+1+j+k}}\ (\to\text{-E1}) \qquad \frac{\Gamma \vdash e_1 : {}_{m}\tau_1 \longrightarrow \tau_2{}^{\perp}_{n+1} \qquad \Gamma \vdash e_2 : {}_{m+1}\tau_1{}^{\dagger}_{n}}{\Gamma \vdash e_1\textvisiblespace e_2 : {}_{m}\tau_2{}^{\dagger}_{n+1}}\ (\to\text{-E2})$$

$$\frac{\Gamma \vdash e_1 : {}_{m}\tau_1 \longrightarrow \tau_2 l\Phi^{\dagger}_{n+1} \qquad \Gamma \vdash e_2 : {}_{m+1}\tau_1{}^{\top}_{n}}{\Gamma \vdash e_1\textvisiblespace e_2 : {}_{m}\tau_2{}^{\dagger}_{n+1}}\ (\to\text{-E3}) \qquad \frac{\Gamma \vdash e_1 : {}_{m}\tau_1{}^{\emptyset}_{n} \qquad \Gamma \vdash e_2 : {}_{j}\tau_2{}^{\emptyset}_{k}}{\Gamma \vdash (e_1, e_2){}_{0}\tau_1 \times \tau_2{}^{\emptyset}_{m+n+j+k+3} :}\ (\times\text{-I1}) \qquad \frac{\Gamma \vdash e_1 : {}_{m}\tau_1{}^{\dagger}_{n} \qquad \Gamma \vdash e_2 : {}_{m+1}\tau_2{}^{\top}_{n}}{\Gamma \vdash \mathsf{true} : {}_{0}\tau_1 \times \tau_2{}^{\dagger}_{m+n+1}}\ (\times\text{-I2})$$

$$\frac{\Gamma \vdash e_1 : {}_{m}\tau_1 \times \tau_2{}^{\emptyset}_{n} \qquad \Gamma, x:\tau_1, y:\tau_2 \vdash e_2 : {}_{j+1}\tau'^{\emptyset}_{k}}{\Gamma \vdash \mathsf{let}\ (x, \textvisiblespace y) = e_1\ \mathsf{in}\ e_2 : {}_{0}\tau'^{\emptyset}_{5+|x|+2+|y|+3+m+n+3+j+1+k}}\ (\times\text{-E1}) \qquad \frac{\Gamma \vdash e_1 : {}_{m}\tau_1 \times \tau_2{}^{\emptyset}_{n} \qquad \Gamma, x:\tau_1, y:\tau_2 \vdash e_2 : {}_{j+1}\tau'^{\mathrm{I}}_{k}}{\Gamma \vdash \mathsf{let}\ (x, \textvisiblespace y) = e_1\ \mathsf{in}\ e_2 : {}_{0}\tau'^{\dagger}_{\max(5+|x|+2+|y|+3+m+n+3,j+1+k)}}\ (\times\text{-E2})$$

$$\frac{\Gamma \vdash e_1 : {}_{m+1}\tau_1 \times \tau_2{}^{\mathrm{I}}_{n} \qquad \Gamma, x:\tau_1, y:\tau_2 \vdash e_2 : {}_{m+1}\tau'^{\mathrm{I}}_{n}}{\Gamma \vdash \mathsf{let}\ (x, \textvisiblespace y) = e_1\ \mathsf{in}\ e_2 : {}_{0}\tau'^{\dagger}_{\max(5+|x|+2+|y|+3,\, m+1+n)}}\ (\times\text{-E3}) \qquad \frac{\Gamma \vdash e_1 : {}_{m+1}\mathsf{bool}^{\emptyset}_{n} \qquad \Gamma \vdash e_2 : {}_{j+1}\tau^{\emptyset}_{k} \qquad \Gamma \vdash e_3 : {}_{i+1}\tau^{\emptyset}_{l}}{\Gamma \vdash \mathsf{if}\ e_1\textvisiblespace\mathsf{then}\ e_2\textvisiblespace\mathsf{else}\ e_3 : {}_{0}\tau^{\emptyset}_{2+m+1+n+5+j+1+k+5+i+1+l}}\ (\text{bool-E1})$$

$$\frac{\Gamma \vdash e_1 : {}_{m+1}\mathsf{bool}^{\emptyset}_{n} \qquad \Gamma \vdash e_2 : {}_{j+1}\tau^{\mathrm{I}}_{k} \qquad \Gamma \vdash e_3 : {}_{j+1}\tau^{\top}_{k}}{\Gamma \vdash \mathsf{if}\ e_1\textvisiblespace\mathsf{then}\ e_2\textvisiblespace\mathsf{else}\ e_3 : {}_{0}\tau^{\dagger}_{\max(2+m+1+n,j+1+k)}}\ (\text{bool-E2}) \qquad \frac{\Gamma \vdash e_1 : {}_{m+1}\mathsf{bool}^{\mathrm{I}}_{n} \qquad \Gamma \vdash e_2 : {}_{m+1}\tau^{\mathrm{I}}_{n} \qquad \Gamma \vdash e_3 : {}_{m+1}\tau^{\top}_{n}}{\Gamma \vdash \mathsf{if}\ e_1\textvisiblespace\mathsf{then}\ e_2\textvisiblespace\mathsf{else}\ e_3 : {}_{0}\tau^{\dagger}_{\max(4,m+1+n)}}\ (\text{bool-E3})$$

$$\frac{\Gamma \vdash e_1 : {}_{m}\tau_1{}^{\emptyset}_{n} \qquad \Gamma, x:\tau_1 \vdash e_2 : {}_{j+1}\tau_2{}^{\emptyset}_{k}}{\Gamma \vdash \mathsf{let}\textvisiblespace x\textvisiblespace = e_1\textvisiblespace\mathsf{in}\ e_2 : {}_{0}\tau'^{\emptyset}_{4+|x|+2+m+n+3+j+1+k}}\ (\textsc{Let1}) \qquad \frac{\Gamma \vdash e_1 : {}_{m}\tau_1{}^{\emptyset}_{n} \qquad \Gamma, x:\tau_1 \vdash e_2 : {}_{j+1}\tau_2{}^{\mathrm{I}}_{k}}{\Gamma \vdash \mathsf{let}\textvisiblespace x\textvisiblespace = e_1\textvisiblespace\mathsf{in}\ e_2 : {}_{0}\tau'^{\dagger}_{\max(4+|x|+2+m+n+3,j+1+k)}}\ (\textsc{Let2})$$

$$\frac{\Gamma \vdash e_1 : {}_{m+1}\tau_1{}^{\mathrm{I}}_{n} \qquad \Gamma, x:\tau_1 \vdash e_2 : {}_{m+1}\tau_2{}^{\mathrm{I}}_{n}}{\Gamma \vdash \mathsf{let}\textvisiblespace x\textvisiblespace = e_1\textvisiblespace\mathsf{in}\ e_2 : {}_{0}\tau'^{\dagger}_{\max(4+|x|+2,m+1+n)}}\ (\textsc{Let3}) \qquad \frac{\Gamma \vdash e : {}_{m}\tau^{\emptyset}_{n}}{\Gamma \vdash \textvisiblespace e : {}_{m+1}\tau^{\emptyset}_{n}}\ (\textsc{Space1}) \qquad \frac{\Gamma \vdash e : {}_{m}\tau^{\emptyset}_{n}}{\Gamma \vdash e\textvisiblespace : {}_{m}\tau^{\emptyset}_{n+1}}\ (\textsc{Space2}) \qquad \frac{\Gamma \vdash e : {}_{m}\tau^{\dagger}_{n}}{\Gamma \vdash \backslash\mathsf{n}e : {}_{m}\tau^{\top}_{n}}\ (\textsc{Newline1})$$

$$\frac{\Gamma \vdash e : {}_{m}\tau^{\perp}_{n}}{\Gamma \vdash \backslash\mathsf{n}e : {}_{m}\tau^{\mathrm{I}}_{n}}\ (\textsc{Newline2}) \qquad \frac{\Gamma \vdash e : {}_{m}\tau^{\dagger}_{n}}{\Gamma \vdash e\backslash\mathsf{n} : {}_{m}\tau^{\perp}_{n}}\ (\textsc{Newline3}) \qquad \frac{\Gamma \vdash e : {}_{m}\tau^{\top}_{n}}{\Gamma \vdash e\backslash\mathsf{n} : {}_{m}\tau^{\mathrm{I}}_{n}}\ (\textsc{Newline2}) \qquad \frac{\Gamma \vdash e : {}_{m}\tau^{\nu}_{n_1} \qquad n_1 \leq n_2}{\Gamma \vdash e : {}_{m}\tau^{\nu}_{n_2}}\ (\textsc{Sub1})$$

$$\frac{\Gamma \vdash e : {}_{m}\tau^{\emptyset}_{n}}{\Gamma \vdash e : {}_{m}\tau^{\dagger}_{n}}\ (\textsc{Sub2})$$

Figure 4: Statics of Pedant

none of the three subexpressions contain newlines, and therefore the entire if statement appears on one line:

```
if b then f () else g ()
```

All three subexpressions must begin with at least one space, in order to separate the expression from the preceding keyword. This is enforced with the +1 on the left component of the type of each subexpression. Rule bool-E2 assumes that the condition has no newlines, but requires the "then" branch to have starting and ending newlines and the "else" branch to have a top newline. As before, all subexpressions must begin with whitespace. Now, however, this results in requiring at least one space of indentation for the two branches, as both are required to begin with a newline.

```
if b then
  f ()
else
  g ()
```

Finally, rule bool-E3 additionally assumes that the condition has starting and ending newlines.

```
if b then f ()
else g ()
```

```
if b then
  f ()
else g ()
```

```
if
b then
  f ()
else g ()
```

Figure 5: Badly formatted expressions that can't typecheck in Pedant.

```
if
  b
then
  f ()
else
  g ()
```

Note that because these are the only three typing rules for "if", none of the abominations in Figure 5 can typecheck.

In all cases, the left column of the if expression is 0, because the expression does not start with whitespace. The width is calculated appropriately from the widths of the subexpressions and keywords. For multiline expressions, we take the maximum width. Note that in bool-E2, for example, both branches are assigned the same width. To make this work, we allow for a form of subtyping (Sub1) which we call *width subtyping*, a name so well-suited to this concept that I won't bother Googling for whether or not it's already used for a different concept. This allows a narrower expression to be assigned a wider type for the purposes of taking the maximum width of several subexpressions. We also allow for another form of subtyping on line breaks (Sub2) in which "none" is considered a subtype of "internal". For this, we will use the similarly well-suited term *depth subtyping*.

At this point in a type systems paper, we would normally define a dynamic semantics for the language, prove progress and preservation and declare success. The astute reader may notice, however, that the type system we have defined fails at even the most basic aspects of being a type system. In fact, not only would preservation be false for any reasonable dynamic semantics, but we can't even prove a reasonable substitution lemma:

$$x : \mathsf{int} \vdash x : {}_0\mathsf{int}_1^{\emptyset}$$

$$[42/x]x = 42$$

$$\cdot \vdash 42 : {}_0\mathsf{int}_2^{\emptyset}$$

At this point, we have several options:

(1) We could admit that this whole idea is a farce and these are entirely syntactic properties that we have no business trying to enforce semantically, and scrap this idea altogether.
(2) We could quietly admit the above but recognize that this whole conference is a farce and decide that it doesn't matter.
(3) We could double down and go to even more ridiculous lengths to try to make this idea at least *seem* reasonable.

For maximum comedic effect, we will, of course, proceed with Option 3.

```
let c = "SIGBOVIK␣2022" in
let how = "really␣really␣really␣" in
let what = "bad␣idea" in
"This␣" ^ c ^ "␣paper␣is␣a␣" ^ how ^ what

let how = "really␣really␣really␣" in
let what = "bad␣idea" in
"This␣" ^ "SIGBOVIK␣2022" ^ "␣paper␣is␣a␣" ^ how ^ what

let what = "bad␣idea" in
"This␣" ^ "SIGBOVIK␣2022" ^ "␣paper␣is␣a␣" ^ "really␣really␣really␣" ^

...
```

Figure 6: A single-step execution of an OCaml expression.

3 THE LANGUAGE PRETTYPRINT

Before attempting to make the ideas of the previous section kinda sorta work, we need a flimsy justification for doing so. Most functional languages are presented with an operational semantics that involves transforming expressions (e.g. applying substitutions, reducing "if" statements when the condition is evaluated), as opposed to imperative languages, which are generally presented as static code with a program counter that captures the runtime control flow. This poses a problem for debuggers for functional languages: while debuggers for imperative languages can easily show the line of source code corresponding to the point of execution and display the values of variables, debuggers for functional languages are generally not able to do something immediately analogous. Part of the problem is that functional languages are generally not actually evaluated in a way that closely resembles the abstract operational semantics presentation: they are either changed heavily during compilation or interpreted using more efficient techniques. However, one could imagine building an educational debugger for a functional language that allows novice functional programmers to single-step programs in a way that follows the formal operational semantics they learned for the language. This then presents the problem of displaying the program at any point during execution. One option would be to maintain the AST of the program as it is transformed by execution, and pretty-print it when needed. However, this would obscure many of the transformations. Statements that had been spread across many lines might now be condensed into one or vice versa, and it would be difficult for novices to follow how the code moves across the screen. It would be preferable for this contrived example if the line breaks in the original code were maintained, so the steps were clear. But then, with no restrictions on input programs, it would be possible for some of these intermediate expressions to be too wide to properly display. Consider the example in Figure 6.

This flimsy justification gives us a new, and actually less ridiculous, interpretation for the types of expressions: the type should describe the maximum width that an expression will have during execution, rather than simply the width of the source expression. This now begins to more closely resemble an *effect* or *type-and-effect* system. As with most effect systems, this requires us to annotate more types. For example, it is now no longer sufficient to describe a function type by its input and output base types, its left column and its width. Effect systems for call-by-value languages would

affect
verb tr.

(1) (of things) to tend toward habitually or naturally.
(2) to assume artificially, pretentiously, or for effect: *to affect a Southern accent.*

Figure 7: Definitions of *affect*, paraphrased from Dictionary.com [1]

generally annotate the function type (or possibly the return type, depending on the desired notation) with the effects that might occur during execution of the function. We could similarly annotate the return type of a function with a left, width and newline description, indicating the behavior of the function as it executes. However, this will depend on the width and line breaks of the substituted arguments. Consider the following function:

```
fun x -> "Is␣this␣too␣long?␣It␣depends␣on␣" ^ x
```

So we must also annotate the domains of functions with their width and line breaks (it will turn out not to be necessary to annotate the left of domains because of how we design the dynamics). This is not generally the case in effect systems for call-by-value languages because passed arguments will be values, which will by definition have no effects. We observe, however that the width of an expression is not really an effect, but rather an *innate property* of the expression which we must consider for both values and non-value expressions. Because of this, we refer to our novel construction as a *type-and-affect* system (see Figure 7).

In keeping with our new motivation, we will call this language PrettyPrint. The revised statics appear in Figure 8. For the most part, the rules are similar to those for Pedant. In addition to annotating the types of codomains and domains of functions (and the components of product types), we now annotate variables in the context with their width and newlines. The variable rule then assigns a variable x the maximum of $|x|$ and the width from the context. The function application rules ensure that function arguments match the function's expected width and newlines, with the caveat that because of depth subtyping, arguments with no line breaks, i.e., arguments written in one row of text, can be passed to functions expecting multi-row "internal" arguments, allowing us a form of *row polymorphism* (again, no Googling necessary).

Finally, we present the dynamics of PrettyPrint in Figure 9. The dynamics depend on two auxiliary definitions, *strip*(e) and $\overleftarrow{e}$, defined in Figure 10. The function *strip*(e) strips leading and trailing whitespace from an expression. The function $\overleftarrow{e}$ removes indentation from e. It is defined in terms of *Indentation*(e), which calculates the indentation level of e. We remove indentation and trailing and leading whitespace before performing substitution. We must also look past whitespace to perform reductions. Otherwise, the semantics are standard.

We will now state, without any attempt at proof, several facts about the correctness of these operations that are probably at least close to true.

Lemma 1. *If* $\Gamma \vdash e : {}_{m}\tau\frac{\nu}{n}$ *then* $\Gamma \vdash \overleftarrow{e} : {}_{0}\tau\frac{\nu}{n-Indentation(e)}$.

Lemma 2. *If* $\Gamma \vdash e : {}_{m}\tau\frac{\nu}{n}$ *then* $\Gamma \vdash strip(e) : {}_{m}\tau^{\dagger}_{n}$. *If* $\Gamma \vdash e : {}_{m}\tau^{\emptyset}_{n}$ *then* $\Gamma \vdash strip(e) : {}_{m}\tau^{\emptyset}_{n}$.

Lemma 3 (Substitution). *If* $\Gamma, v : \tau 1\frac{\nu_1}{k} \vdash e : {}_{m}\tau 2\frac{\nu_2}{n}$ *and* $\Gamma \vdash v : {}_{j}\tau 1\frac{\nu_1}{k}$, *then* $\Gamma \vdash e[strip(\overleftarrow{v})/x] : {}_{m}\tau 2\frac{\nu_2}{n}$.

Finally, we can state and not attempt to prove type safety.

Theorem 1 (Preservation). *If* $\bullet \vdash e : {}_{m}\tau\frac{\nu}{n}$ *and* $e \longrightarrow e'$ *then* $\bullet \vdash e' : {}_{m}\tau\frac{\nu}{n}$.

Of course, the canonical forms lemma now becomes interesting because irreducible values may have leading or trailing whitespace.

Lemma 4 (Canonical Forms).

(1) *If* $\bullet \vdash v : {}_{m}\mathsf{int}\frac{\nu}{n}$, *then* $strip(v) = \overline{n}$ *for some* n.
(2) *If* $\bullet \vdash v : {}_{m}\mathsf{bool}\frac{\nu}{n}$, *then* $strip(v) = \mathsf{true}$ *or* $strip(v) = \mathsf{false}$.
(3) *If* $\bullet \vdash v : {}_{m}\tau_1 \longrightarrow \tau 2\frac{\nu}{n}$, *then* $strip(v) = \mathsf{fun}␣x␣{-}{>}\ e$.
(4) *If* $\bullet \vdash v : {}_{l\,m}\tau 1\frac{\nu_1}{n} \times {}_{j}\tau 2\frac{\nu_2}{k}\frac{\nu}{w}$ *then* $strip(v) = (v_1, v_2)$.

Theorem 2 (Progress). *If* $\bullet \vdash e : {}_{m}\tau\frac{\nu}{n}$ *then* e *is a value or there exists* e' *such that* $e \longrightarrow e'$.

4 IMPLEMENTATION

Yes, you read that section header correctly. This paper is not a joke. Well, it is. But it's a joke to which the author is deeply, deeply committed for reasons that are not clear in the slightest. So yeah, we implemented a mostly[2]-working parser, type checker and single-step interpreter for PrettyPrint.

Type inference is about as awful as you'd expect. The type system is too weird for standard unification algorithms, so instead, the implementation traverses the program and generates a set of constraints on the width, left and newline behavior of each expression. At the end, a constraint is added restricting the overall width of the expression to be, of course, 80. We then solve these constraints using Z3 [5]. Note that the constraints on the width and left form an integer linear program (ILP). We leave it to future work to determine whether arbitrary ILPs can be encoded as PrettyPrint programs, making typechecking NP-complete.

After the program is rejected, revised, rejected again, and revised a few more times, and finally typechecked, the completed program is displayed and the user given the option to step the program one step at a time or to completion, as shown in Figure 11. This paper has already gone on too long, so rather than further bore you with details of the implementation, we'll give a few observations about our experience doing the implementation in a nice, easy-to-digest list:

- As the name of the language suggests, pretty-printing, which, frankly, I've always found to be the most difficult part of language implementation, is now trivial.
- Lexing, typically the most trivial part of language implementation, is now even more trivial as you don't even need to figure out how to skip over whitespace.

[2] well, somewhat

$$\frac{}{\Gamma, x : \tau^{\nu}_{n} \vdash x : {}_{0}\tau^{\nu}_{\max(n,|x|)}}\ (\textsc{Var}) \qquad \frac{}{\Gamma \vdash \overline{n} : {}_{0}\mathsf{int}^{\emptyset}_{|\overline{n}|}}\ (\text{int-I}) \qquad \frac{}{\Gamma \vdash \mathsf{true} : {}_{0}\mathsf{bool}^{\emptyset}_{4}}\ (\text{bool-I1}) \qquad \frac{}{\Gamma \vdash \mathsf{false} : {}_{0}\mathsf{bool}^{\emptyset}_{5}}\ (\text{bool-I2})$$

$$\frac{\Gamma, x : \tau_1 \vdash e : {}_{m+1}\tau_2{}^{\emptyset}_{n}}{\Gamma \vdash \mathsf{fun}\text{␣}x\text{␣}\mathsf{->}\ e : {}_{0}\tau_1 \to \tau_2{}^{\emptyset}_{4+|x|+4+m+n}}\ (\to\text{-I1}) \qquad \frac{\Gamma, x : \tau_1 \vdash e : {}_{m+1}\tau_2{}^{\top}_{n}}{\Gamma \vdash \mathsf{fun}\text{␣}x\text{␣}\mathsf{->}\ e : {}_{0}\tau_1 \to \tau_2{}^{\dagger}_{\max(m+1+n,4+|x|+4)}}\ (\to\text{-I2})$$

$$\frac{\Gamma \vdash e_1 : {}_{m}\tau_1{}^{\emptyset}_{k} \to \tau_2{}^{\emptyset}_{n} \qquad \Gamma \vdash e_2 : {}_{j}\tau_1{}^{\emptyset}_{k}}{\Gamma \vdash e_1\text{␣}e_2 : {}_{m}\tau_2{}^{\nu}_{n+1+j+k}}\ (\to\text{-E1}) \qquad \frac{\Gamma \vdash e_1 : {}_{m}\tau_1{}^{\dagger}_{n} \to \tau_2{}^{\perp}_{n+1} \qquad \Gamma \vdash e_2 : {}_{m+1}\tau_1{}^{\dagger}_{n}}{\Gamma \vdash e_1\text{␣}e_2 : {}_{m}\tau_2{}^{\dagger}_{n+1}}\ (\to\text{-E2})$$

$$\frac{\Gamma \vdash e_1 : {}_{m}\tau_1{}^{\dagger}_{n} \to \tau_2{}^{\dagger}_{n+1} \qquad \Gamma \vdash e_2 : {}_{m+1}\tau_1{}^{\top}_{n}}{\Gamma \vdash e_1\text{␣}e_2 : {}_{m}\tau_2{}^{\dagger}_{n+1}}\ (\to\text{-E3}) \qquad \frac{\Gamma \vdash e_1 : {}_{m}\tau_1{}^{\emptyset}_{n} \qquad \Gamma \vdash e_2 : {}_{j}\tau_2{}^{\emptyset}_{k}}{\Gamma \vdash (e_1, e_2) : {}_{0}\tau_1 \times \tau_2{}^{\emptyset}_{m+n+j+k+3}}\ (\times\text{-I1}) \qquad \frac{\Gamma \vdash e_1 : {}_{m}\tau_1{}^{\dagger}_{n} \qquad \Gamma \vdash e_2 : {}_{m+1}\tau_2{}^{\top}_{n}}{\Gamma \vdash (e_1, e_2) : {}_{0}\tau_1 \times \tau_2{}^{\dagger}_{m+n+1}}\ (\times\text{-I2})$$

$$\frac{\Gamma \vdash e_1 : {}_{m}\left(\tau_1{}^{\nu_1}_{w_1} \times \tau_2{}^{\nu_2}_{w_2}\right)^{\emptyset}_{n} \qquad \Gamma, x : \tau_1{}^{\nu_1}_{w_1}, y : \tau_2{}^{\nu_2}_{w_2} \vdash e_2 : {}_{j+1}\tau'^{\emptyset}_{k}}{\Gamma \vdash \mathsf{let}\ (x,\text{␣}y) = e_1\ \mathsf{in}\ e_2 : {}_{0}\tau'^{\emptyset}_{5+|x|+2+|y|+3+m+n+3+j+1+k}}\ (\times\text{-E1})$$

$$\frac{\Gamma \vdash e_1 : {}_{m}\left(\tau_1{}^{\nu_1}_{w_1} \times \tau_2{}^{\nu_2}_{w_2}\right)^{\emptyset}_{n} \qquad \Gamma, x : \tau_1{}^{\nu_1}_{w_1}, y : \tau_2{}^{\nu_2}_{w_2} \vdash e_2 : {}_{j+1}\tau'^{\mathrm{I}}_{k}}{\Gamma \vdash \mathsf{let}\ (x,\text{␣}y) = e_1\ \mathsf{in}\ e_2 : {}_{0}\tau'^{\dagger}_{\max(5+|x|+2+|y|+3+m+n+3,j+1+k)}}\ (\times\text{-E2})$$

$$\frac{\Gamma \vdash e_1 : {}_{m+1}\left(\tau_1{}^{\nu_1}_{w_1} \times \tau_2{}^{\nu_2}_{w_2}\right)^{\mathrm{I}}_{n} \qquad \Gamma, x : \tau_1{}^{\nu_1}_{w_1}, y : \tau_2{}^{\nu_2}_{w_2} \vdash e_2 : {}_{m+1}\tau'^{\mathrm{I}}_{n}}{\Gamma \vdash \mathsf{let}\ (x,\text{␣}y) = e_1\ \mathsf{in}\ e_2 : {}_{0}\tau'^{\dagger}_{\max(5+|x|+2+|y|+3,m+1+n)}}\ (\times\text{-E3})$$

$$\frac{\Gamma \vdash e_1 : {}_{m+1}\mathsf{bool}^{\emptyset}_{n} \qquad \Gamma \vdash e_2 : {}_{j+1}\tau^{\emptyset}_{k} \qquad \Gamma \vdash e_3 : {}_{i+1}\tau^{\emptyset}_{l}}{\Gamma \vdash \mathsf{if}\ e_1\text{␣}\mathsf{then}\ e_2\text{␣}\mathsf{else}\ e_3 : {}_{0}\tau^{\emptyset}_{2+m+1+n+5+j+1+k+5+i+1+l}}\ (\text{bool-E1}) \qquad \frac{\Gamma \vdash e_1 : {}_{m+1}\mathsf{bool}^{\emptyset}_{n} \qquad \Gamma \vdash e_2 : {}_{j+1}\tau^{\mathrm{I}}_{k} \qquad \Gamma \vdash e_3 : {}_{j+1}\tau^{\top}_{k}}{\Gamma \vdash \mathsf{if}\ e_1\text{␣}\mathsf{then}\ e_2\text{␣}\mathsf{else}\ e_3 : {}_{0}\tau^{\dagger}_{\max(2+m+1+n,j+1+k)}}\ (\text{bool-E2})$$

$$\frac{\Gamma \vdash e_1 : {}_{m+1}\mathsf{bool}^{\mathrm{I}}_{n} \qquad \Gamma \vdash e_2 : {}_{m+1}\tau^{\mathrm{I}}_{n} \qquad \Gamma \vdash e_3 : {}_{m+1}\tau^{\dagger}_{n}}{\Gamma \vdash \mathsf{if}\ e_1\text{␣}\mathsf{then}\ e_2\text{␣}\mathsf{else}\ e_3 : {}_{0}\tau^{\dagger}_{\max(4,m+1+n)}}\ (\text{bool-E3}) \qquad \frac{\Gamma \vdash e_1 : {}_{m}\tau_1{}^{\emptyset}_{n} \qquad \Gamma, x : \tau_1{}^{\emptyset}_{n} \vdash e_2 : {}_{j+1}\tau_2{}^{\emptyset}_{k}}{\Gamma \vdash \mathsf{let}\text{␣}x\text{␣} = e_1\text{␣}\mathsf{in}\ e_2 : {}_{0}\tau'^{\emptyset}_{4+|x|+2+m+n+3+j+1+k}}\ (\textsc{Let1})$$

$$\frac{\Gamma \vdash e_1 : {}_{m}\tau_1{}^{\emptyset}_{n} \qquad \Gamma, x : \tau_1{}^{\emptyset}_{n} \vdash e_2 : {}_{j+1}\tau_2{}^{\mathrm{I}}_{k}}{\Gamma \vdash \mathsf{let}\text{␣}x\text{␣} = e_1\text{␣}\mathsf{in}\ e_2 : {}_{0}\tau'^{\dagger}_{\max(4+|x|+2+m+n+3,j+1+k)}}\ (\textsc{Let2}) \qquad \frac{\Gamma \vdash e_1 : {}_{m+1}\tau_1{}^{\mathrm{I}}_{n} \qquad \Gamma, x : \tau_1{}^{\mathrm{I}}_{n} \vdash e_2 : {}_{m+1}\tau_2{}^{\mathrm{I}}_{n}}{\Gamma \vdash \mathsf{let}\text{␣}x\text{␣} = e_1\text{␣}\mathsf{in}\ e_2 : {}_{0}\tau'^{\dagger}_{\max(4+|x|+2,m+1+n)}}\ (\textsc{Let3})$$

$$\frac{\Gamma \vdash e : {}_{m}\tau^{\emptyset}_{n}}{\Gamma \vdash \text{␣}e : {}_{m+1}\tau^{\emptyset}_{n}}\ (\textsc{Space1}) \qquad \frac{\Gamma \vdash e : {}_{m}\tau^{\emptyset}_{n}}{\Gamma \vdash e\text{␣} : {}_{m}\tau^{\emptyset}_{n+1}}\ (\textsc{Space2}) \qquad \frac{\Gamma \vdash e : {}_{m}\tau^{\dagger}_{n}}{\Gamma \vdash \backslash\mathsf{n}e : {}_{m}\tau^{\top}_{n}}\ (\textsc{Newline1}) \qquad \frac{\Gamma \vdash e : {}_{m}\tau^{\perp}_{n}}{\Gamma \vdash \backslash\mathsf{n}e : {}_{m}\tau^{\mathrm{I}}_{n}}\ (\textsc{Newline2})$$

$$\frac{\Gamma \vdash e : {}_{m}\tau^{\dagger}_{n}}{\Gamma \vdash e\backslash\mathsf{n} : {}_{m}\tau^{\perp}_{n}}\ (\textsc{Newline3}) \qquad \frac{\Gamma \vdash e : {}_{m}\tau^{\top}_{n}}{\Gamma \vdash e\backslash\mathsf{n} : {}_{m}\tau^{\mathrm{I}}_{n}}\ (\textsc{Newline2}) \qquad \frac{\Gamma \vdash e : {}_{m}\tau^{\nu}_{n_1} \qquad n_1 \leq n_2}{\Gamma \vdash e : {}_{m}\tau^{\nu}_{n_2}}\ (\textsc{Sub1}) \qquad \frac{\Gamma \vdash e : {}_{m}\tau^{\emptyset}_{n}}{\Gamma \vdash e : {}_{m}\tau^{\dagger}_{n}}\ (\textsc{Sub2})$$

Figure 8: Statics of PrettyPrint

- Producing reasonable type error messages, possibly the second most difficult part of language implementation behind pretty printing, is actually a little less painful than you might expect for this language. We associate semantic and position information with each constraint passed to Z3. If the constraints are unsatisfiable, we look up this information for

$$\frac{e_1 \rightarrow e_1'}{e_1\text{\textvisiblespace}e_2 \rightarrow e_1'\text{\textvisiblespace}e_2} \qquad \frac{e_2' \rightarrow e_2'}{v_1\text{\textvisiblespace}e_2 \rightarrow v_1\text{\textvisiblespace}e_2'} \qquad \frac{\mathit{strip}(v_1) = \mathsf{fun}\text{\textvisiblespace}x\text{\textvisiblespace}\mathsf{->}\ e}{v_1\text{\textvisiblespace}v_2 \rightarrow e[\mathit{strip}(\overleftarrow{v_2})/x]} \qquad \frac{e_1 \rightarrow e_1'}{(e_1, e_2) \rightarrow (e_1', e_2)} \qquad \frac{e_2' \rightarrow e_2'}{(v_1, e_2) \rightarrow (v_1, e_2')}$$

$$\frac{e_1 \rightarrow e_1'}{\mathsf{let}\ (x, \text{\textvisiblespace}y) = e_1\ \mathsf{in}\ e_2 \rightarrow \mathsf{let}\ (x, \text{\textvisiblespace}y) = e_1'\ \mathsf{in}\ e_2} \qquad \frac{}{\mathsf{let}\ (x, \text{\textvisiblespace}y) = (v_1, v_2)\ \mathsf{in}\ e_2 \rightarrow e_2[\mathit{strip}(\overleftarrow{v_1})/x][\mathit{strip}(\overleftarrow{v_2})/y]}$$

$$\frac{e_1 \rightarrow e_1'}{\mathsf{let}\text{\textvisiblespace}x\text{\textvisiblespace} = e_1\text{\textvisiblespace}\mathsf{in}\ e_2 \rightarrow \mathsf{let}\text{\textvisiblespace}x\text{\textvisiblespace} = e_1'\text{\textvisiblespace}\mathsf{in}\ e_2} \qquad \frac{}{\mathsf{let}\text{\textvisiblespace}x\text{\textvisiblespace} = v\text{\textvisiblespace}\mathsf{in}\ e_2 \rightarrow e_2[\mathit{strip}(\overleftarrow{v})/x]} \qquad \frac{e_1 \rightarrow e_1'}{\mathsf{if}\ e_1\text{\textvisiblespace}\mathsf{then}\ e_2\text{\textvisiblespace}\mathsf{else}\ e_3 \rightarrow \mathsf{if}\ e_1'\text{\textvisiblespace}\mathsf{then}\ e_2\text{\textvisiblespace}\mathsf{else}\ e_3}$$

$$\frac{\mathit{strip}(v) = \mathsf{true}}{\mathsf{if}\ v\text{\textvisiblespace}\mathsf{then}\ e_2\text{\textvisiblespace}\mathsf{else}\ e_3 \rightarrow e_2} \qquad \frac{\mathit{strip}(v) = \mathsf{false}}{\mathsf{if}\ v\text{\textvisiblespace}\mathsf{then}\ e_2\text{\textvisiblespace}\mathsf{else}\ e_3 \rightarrow e_3} \qquad \frac{e \rightarrow e'}{\text{\textvisiblespace}e \rightarrow \text{\textvisiblespace}e'} \qquad \frac{e \rightarrow e'}{e\text{\textvisiblespace} \rightarrow e'\text{\textvisiblespace}} \qquad \frac{e \rightarrow e'}{\backslash\mathsf{n}e \rightarrow \backslash\mathsf{n}e'} \qquad \frac{e \rightarrow e'}{e\backslash\mathsf{n} \rightarrow e'\backslash\mathsf{n}}$$

Figure 9: Dynamics of PRETTYPRINT

each constraint in the unsat core and convert it to halfway-decent error messages.

5 CONCLUSION

I first had the idea for this paper right after SIGBOVIK 2020 (right after SIGBOVIK is, of course, when I have all of my best SIGBOVIK paper ideas). At least, I thought, I had a year to make it actually work out. Of course, I forgot all about this until January 2021 and figured I still had enough time to throw something together. Unfortunately, making this really work, which I was determined to do, took quite a bit more time than that. So, I missed the deadline and figured, well, now I have *another* whole year.

And forgot about it until January 2022. But this time, I managed to scrape it together and submit.

Two years, two core calculi and way too much implementation effort later, we have a dumb language that enforces pedantic whitespace constraints during type checking. Was it worth it? That's for you to decide.

REFERENCES

[1] [n. d.]. Affect. https://www.dictionary.com/browse/affect. Accessed: 3/21/2021.

[2] [n. d.]. Style Guide for C. https://cs50.readthedocs.io/style/c/. Accessed: 3/21/2021.

[3] [n. d.]. Whitespace .NET. Accessed: 3/24/2022.

[4] L.W. Cannon, R.A. Elliott, L.W. Kirchhoff, et al. [n. d.]. Recommended C Style and Coding Standards.

[5] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 337–340.

$$\begin{array}{lcl}
\mathit{strip}(\mathsf{fun}\text{␣}x\text{␣}{-}{>}\ e) & = & \mathsf{fun}\text{␣}x\text{␣}{-}{>}\ \mathit{rstrip}(e) \\
\mathit{strip}(e_1\text{␣}e_2) & = & \mathit{lstrip}(e_1)\text{␣}\mathit{rstrip}(e_2) \\
\mathit{strip}(\mathsf{let}\ (x,\text{␣}y) = e_1\ \mathsf{in}\ e_2) & = & \mathsf{let}\ (x,\text{␣}y) = e_1\ \mathsf{in}\ \mathit{rstrip}(e_2) \\
\mathit{strip}(\mathsf{let}\text{␣}x\text{␣} = e_1\text{␣}\mathsf{in}\ e_2) & = & \mathsf{let}\text{␣}x\text{␣} = e_1\text{␣}\mathsf{in}\ \mathit{rstrip}(e_2) \\
\mathit{strip}(\mathsf{if}\ e_1\text{␣}\mathsf{then}\ e_2\text{␣}\mathsf{else}\ e_3) & = & \mathsf{if}\ e_1\text{␣}\mathsf{then}\ e_2\text{␣}\mathsf{else}\ \mathit{rstrip}(e_3) \\
\mathit{strip}(\text{␣}e) & = & \mathit{strip}(e) \\
\mathit{strip}(e\text{␣}) & = & \mathit{strip}(e) \\
\mathit{strip}(\backslash\mathsf{n}e) & = & \mathit{strip}(e) \\
\mathit{strip}(e\backslash\mathsf{n}) & = & \mathit{strip}(e)
\end{array}$$

$$\begin{array}{lcl}
\mathit{lstrip}(e_1\text{␣}e_2) & = & \mathit{lstrip}(e_1)\text{␣}e_2 \\
\mathit{lstrip}(\text{␣}e) & = & \mathit{lstrip}(e) \\
\mathit{lstrip}(e\text{␣}) & = & \mathit{lstrip}(e)\text{␣} \\
\mathit{lstrip}(\backslash\mathsf{n}e) & = & \mathit{lstrip}(e) \\
\mathit{lstrip}(e\backslash\mathsf{n}) & = & \mathit{lstrip}(e)\backslash\mathsf{n}
\end{array}$$

$$\begin{array}{lcl}
\mathit{rstrip}(\mathsf{fun}\text{␣}x\text{␣}{-}{>}\ e) & = & \mathsf{fun}\text{␣}x\text{␣}{-}{>}\ \mathit{rstrip}(e) \\
\mathit{rstrip}(e_1\text{␣}e_2) & = & e_1\text{␣}\mathit{rstrip}(e_2) \\
\mathit{rstrip}(\mathsf{let}\ (x,\text{␣}y) = e_1\ \mathsf{in}\ e_2) & = & \mathsf{let}\ (x,\text{␣}y) = e_1\ \mathsf{in}\ \mathit{rstrip}(e_2) \\
\mathit{rstrip}(\mathsf{let}\text{␣}x\text{␣} = e_1\text{␣}\mathsf{in}\ e_2) & = & \mathsf{let}\text{␣}x\text{␣} = e_1\text{␣}\mathsf{in}\ \mathit{rstrip}(e_2) \\
\mathit{rstrip}(\mathsf{if}\ e_1\text{␣}\mathsf{then}\ e_2\text{␣}\mathsf{else}\ e_3) & = & \mathsf{if}\ e_1\text{␣}\mathsf{then}\ e_2\text{␣}\mathsf{else}\ \mathit{rstrip}(e_3) \\
\mathit{rstrip}(\text{␣}e) & = & \text{␣}\mathit{rstrip}(e) \\
\mathit{rstrip}(e\text{␣}) & = & \mathit{rstrip}(e) \\
\mathit{rstrip}(\backslash\mathsf{n}e) & = & \backslash\mathsf{n}\mathit{rstrip}(e) \\
\mathit{rstrip}(e\backslash\mathsf{n}) & = & \mathit{rstrip}(e)
\end{array}$$

$$\begin{array}{lcl}
\mathit{Indentation}(v) & = & 0 \\
\mathit{Indentation}(\text{␣}e) & = & 1 + \mathit{Indentation}(e) \\
\mathit{Indentation}(e\text{␣}) & = & \mathit{Indentation}(e) \\
\mathit{Indentation}(\backslash\mathsf{n}e) & = & \mathit{Indentation}(e) \\
\mathit{Indentation}(e\backslash\mathsf{n}) & = & \mathit{Indentation}(e)
\end{array}$$

$$\begin{array}{lcl}
\overleftarrow{\text{␣}e}^{(m+1,n)} & = & \overleftarrow{e}^{(m,n)} \\
\overleftarrow{e\text{␣}}^{(m,n)} & = & \overleftarrow{e}^{(m,n)}\text{␣} \\
\overleftarrow{\backslash\mathsf{n}e}^{(m,n)} & = & \backslash\mathsf{n}\overleftarrow{e}^{(n,n)} \\
\overleftarrow{e\backslash\mathsf{n}}^{(m,n)} & = & \overleftarrow{e}^{(m,n)}\backslash\mathsf{n} \\
\overleftarrow{e} & = & \overleftarrow{e}^{(\mathit{Indentation}(e),\mathit{Indentation}(e))}
\end{array}$$

Figure 10: Auxiliary definitions for dynamics.

```
$ ./pdb prog.prp
Loading prog.prp...
Type checking...
Done.
--------
let x = "I can't believe" in
  let y = "this works" in
    (x, y)
--------
s
--------

  let y = "this works" in
    ("I can't believe", y)
--------
s
--------


    ("I can't believe", "this works")
--------
s
Execution is complete.
--------


    ("I can't believe", "this works")
--------
q
```

Figure 11: Example run of the PrettyPrint interpreter.

br++: A MUCH NEEDED EXTENSION TO brainfuck

CÉDRIC HO THANH

Abstract. TODO: Write the abstract.

1. Introduction

In the year of the lord 2022, Urban Müller's celebrated brainfuck langage is used in numerous industries. For example, my net banking app is written in brainfuck so it's super dooper fast and reliable. It is no surprise then that brainfuck developers are highly sought after in all branches of the software development industry, with an normalized average median income of $0.5.

Nonetheless, after almost 30 years of continued use, brainfuck starts to show its age. Many trendy paradigms and buzzwords are absent from the langage, such as "fullstack", "noSQL", "blockchain", and even "equal pay". In this paper, we introduce brainfuck++ (hereafter br++ for short), an extension that adds modern constructs to brainfuck. While we do not claim to address all the selling points above, we are confident this update lays some robust foundations to tackle them efficiently in ulterior works. Just like we tackle this segway to our sponsors.

Acknowledgments. Thank you to HardBoiledEgg for sponsoring this episode of UNIX Tech Tips. HardBoiledEgg retails all your favorite tech goodies, from old CPUs to faulty motherboards, and offers a premium 24/7 egg yolk-based customer support. They amazing return policy makes sure to call you out when you return a defective product, and even when you don't. So what are you waiting for? Don't let your government have all the fun and start getting egg-screwed today with HardBoiledEgg. Check them out with the link in the description below. You've been reading this entire spot with Linus's voice stop lying.

2. Vanilla extract brainfuck

brainfuck is a surprisingly simple langage. The execution environment (or BVM) consists of an infinite array of 8-bits bytes[1] A called the *tape*, and a data pointer $p \in \mathbb{N}$. The data pointer and the cells of A are initialized to 0. The langage consists of 8 keywords, echoing the bit size of bytes (8) in a heavenly synergy only fathomable by the most neurologically endowed individuals.

Date: March 2022.
Key words and phrases. Programing langage, brainfuck, concurrency, machine learning.

[1] historically, a fixed 30 000-cell array, but memory is cheap nowadays eh?

Keyword	Semantics
`>`	$p \leftarrow p + 1$
`<`	$p \leftarrow p - 1$
`+`	$A_p \leftarrow A_p + 1$
`-`	$A_p \leftarrow A_p - 1$
`.`	Outputs A_p
`,`	Inputs 1 byte into A_p
`[`	Noop woop woop
`]`	If $A_p \neq 0$, then jump to the matching `[`, skip o.w.

If at any point in the execution, the data pointer p becomes negative and < 0, then a `DataPointerUnderflowError` exception is raised. Likewise, if $p \geq \infty$, then a `DataPointerOuttaHereError` exception is raised. It is common practice to raise these twice, in case the first one is lost. For convenience, white spaces, carriage returns and tabulations are accepted by the interpreter, but have a noop woop woop semantics. Technically, non-well bracketed programs are valid, but if a `]` incurs a jump to a non-existing `[`, a generic kernel panic is initiated (twice).

Here is a simple `brainfuck` program that prompts for 2 bytes (or reads them from STDIN) and outputs their sum:

```
, > , [ - > + < ] > .
```

Theorem 2.1. *`brainfuck` is Turing-complete.*

Proof. The following program writes `Turing` on the tape so we should be good.

```
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + > + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + > + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + > + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + > + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + > + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + +
```

□

3. BRAINFUCK++

We now build upon section 2 and provide a complete specification of `br++`. Buckle up pleb.

3.1. Headers.

A `br++` program may start with a *header*, which is just a sequence of keywords specifying various properties and runtime parameters.

- **Bigbyte.** The `B` header keyword specifies that the tape cells are bigbytes instead of traditional 8-bits bytes. A bigbyte is 9-bits big. [2]
- **Root privileges.** The `R` header keyword indicates that the program should run with root privileges. The BVM escalates using standard tools such as dirtyc0w, Pegasus, Dirty pipe, phishing the system administrator, $5 wrench, etc.
- **Archlinux support.** If `A` is specified in the header, the program will output the string `I use arch btw.` (with a newline) everytime the code pointer moves.
- **Non-horizontal semantics.** Instead of being a horizontal array, the `|` transforms the tape into a vertical one. The keywords `>` and `<` are replaced by `^` and `v` respectively.
- **Unicode support.** Classically, `brainfuck` program files are encoded in ASCII. The 8 directive informs the BVM that the file is encoded in UTF8.
- **Online assistance.** The `H` header enables the online assistance facilities. When an exception is raised, the BVM opens StackOverflow and queries the exception type and any accompanying error message.
- **True concurrency.** The `C` header enables true concurrency. See section 3.14.

3.2. **Comments.** Readability is key to write readable code. For that reason, this specification carefully specifies what comments are: they're like in Python. Aditionally, because readability is so crucial to producing high-quality code, the `br++` interpreter will reject programs that do not include at least one *helpful* comment[3], raising a `CodeUnreadableError` exception.

3.3. **Cell packing.** Even with the vast possibilities offered by the potent bigbyte construct, `br++` offers a paradigm to manipulate large data values. This is called *cell packing*. Cell packing is accomplished with the `p` keyword. When the code pointer encounters `p`, the integer value x of the current cell is read, and the following x cells are considered *packed*.

The values of a packed cell is the binary concatenation of all values of all packed cells to its left (or underneath if the tape is vertical). The following example is fairly self-explanatory:

```
+ + +       # |   3 |   0 |   0 |   0 |   0 |
            #     ^
p           # |   3 |   0 :   0 :   0 |   0 |
            #     ^  The next 3 cells are packed
> > >       # |   3 |   0 :   0 :   0 |   0 |
            #                       ^
-           # |   3 | 255 : 255 : 255 |   0 |
            #                       ^
.           # Outputs 2^24 - 1 = 16777215
<           # |   3 | 255 : 255 : 255 |   0 |
            #                 ^
.           # Outputs 2^16 - 1 = 65535
-           # |   3 | 255 : 254 : 255 |   0 |
```

[2] A bigbite in my bigmac is about 25% of the burger, meaning I can eat it in about 4 bigbites. Comment down below with your high score!

[3] The precise semantic of *helpful* is left at the discretion of the implementation.

```
            #                 ^
.           # Outputs 2^16 - 2 = 65534
<           # |   3 | 255 : 254 : 255 |   0 |
            #           ^
.           # Outputs 255
```

If a 0-cell packing is attempted, a `PackingEmptyOniichanNoBakaError` exception is raised.

3.4. **Convention: floating point numbers.** In `br++`, *single precision floating point numbers* are simply sequences of 4 packed cells. For better buoyancy, when working with bigbytes, the high 4 bits of the first cell are considered as padding. For example, the *Nice constant* on tape would look like this `|  66 : 139 :  97 :  72 |`.

3.5. **String literals.** A string literal starts and ends with the keyword `"` and can contain any ASCII character verbatim. The character `"`, however, must be escaped as `\"`, and the backslash `\` by `\\`. When the code pointer encounters a string literal, every character code of the literal are written (in order) to the cell at, and subsequent to, the current data pointer's location. Additionally, a null-terminator is added. The data pointer does not move.

```
            # Initial tape:
            # |   1 |   2 |   3 |   4 |   5 |   6 |
            #     ^
"A\"
b"          #     A     "    \n     b
            # |  65 |  34 |  13 | 100 |   0 |   6 |
            #     ^
```

If the program is UTF8 empowered™, UTF8 strings are accepted. Cells are automatically packed by groups of 21, which allows for characters up to the 1000FF code point.

3.6. **Heap-hop and dynamic mallocation.** With this construct, we aim to mimic the common use of explicitely allocated memory and the heap, as is done with other (inferior) langages such as C.

Dynamic allocation is done with the *malloc* keyword `m`. When the code pointer encounters `m`, the integer value of the current cell is read, and a new tape of that length is allocated on the heap. [4]

3.7. **Program expansion.** A program (and indeed, any finite sequence of integers) can be encoded into a single integer using the following procedure. First, write $p_1, p_2, p_3, \ldots$ for the sequence of prime numbers $2, 3, 5, \ldots$. Then, a sequence $x_1, x_2, x_3, \ldots, x_n \in \mathbb{N}$ (in our case, ASCII character codes) can be encoded as

$$x = \prod_{i=1}^{n} p_i^{x_i},$$

which is oftentimes a big boy number[5]. For example, the sequence $1, 2, 3$ is encoded by $2^1 \times 3^2 \times 5^3 = 2250$, while the string `Hello world!` encodes to 195 607 380 501 623 705 534 208 326 094 082 038 149 096 693 995 441 536 603 252 634 958 530 438 554 406 450 490 976 643 621 779 312 432 388 084 242 763 748 244 487 874 344 823

[4] There is no way to access this new tape or free it.

[5] also called Gödel's number by the mathematical community.

747 974 447 174 082 236 430 459 615 458 978 818 247 973 956 107 539 906 109 369 893 401 173 401 516 997 038 330 150 975 937 285 576 171 852 276 655 063 337 197 226 656 523 237 039 666 508 189 810 223 186 619 820 864 165 434 550 339 911 005 751 544 052 505 591 487 270 778 943 545 799 041 992 230 193 245 034 335 791 651 468 414 607 612 958 810 178 942 795 751 471 227 443 645 921 363 074 110 065 531 215 193 103 128 761 541 211 211 476 319 680 954 901 952 215 014 935 430 435 115 302 455 079 990 811 992 543 403 660 063 804 034 991 262 105 254 957 044 042 231 119 234 784 858 876 374 100 885 283 188 659 170 598 777 349 768 521 363 820 242 235 142 780 890 208 200 255 888 601 216 401 804 150 071 098 331 031 946 232 951 229 284 503 169 196 409 012 301 120 797 494 377 679 005 968 502 505 964 768 810 320 478 250 053 565 749 756 760 163 748 589 560 141 220 013 934 749 208 175 929 238 871 907 913 478 615 045 908 922 992 707 349 525 331 334 708 779 423 469 086 418 674 434 586 265 480 097 791 064 257 314 816 771 165 209 849 320 015 628 002 193 854 527 444 945 885 431 678 843 218 561 565 116 147 760 578 584 267 191 138 303 463 674 130 036 052 448 832 551 754 215 761 591 855 818 729 203 425 917 432 460 982 047 791 080 335 919 245 709 584 281 926 468 086 157 915 818 734 190 653 529 367 409 646 511 077 880 859 375 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 reversed: x_i is the greatest integer such that x is divisible by $p_i^{x_i}$.

Any ASCII string can be encoded and decoded using these methods, and in particular, so can br++ programs. The æ keyword does exactly that. When encountered, the value of the current cell is decoded into a br++ program. The data pointer is incremented, and the decoded program is executed. When it terminates, the execution of the current program resumes.

Here is an example. Remember that the following concise br++ program adds the values of the first two cells and writes the result to the second cell:

```
[ - > + < ]
```

Removing spaces, the encoding of this program is 4 167 109 678 750 440 801 834 791 326 555 089 952 769 865 994 828 551 024 009 245 204 715 556 129 614 039 321 687 018 746 602 136 184 526 862 195 689 060 760 391 695 350 607 989 014 685 486 097 422 187 110 755 435 087 977 407 298 280 835 543 147 132 224 053 643 001 830 439 539 087 674 077 039 427 632 653 926 400 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000. Now, if one wants to compute $1+1$ in a distinguished manner, here is a way:

```
            # We first pack 123 cells
"{"p        # | 123 |   0 :   0 :   0 :   0 : ...
            #    ^
+ + + ...   # We then write 41671... in the next 123 packed cells
            # | 123 |   6 : 134 :  81 :  96 : ... :   0 |
            #                                         ^
> + > +     # We now write the program's inputs
< <         # | 123 |   6 : 134 :  81 :  96 : ... :   0 |   1 |   1 |
            #                                         ^
æ           # | 123 |   6 : 134 :  81 :  96 : ... :   0 |   0 |   2 |
            #                                                     ^
```

3.8. **Strong typing.** [6]

[6] I don't think so.

3.9. **File handling.** Reading a file is accomplished using the `r` keyword. When the code pointers encounters it, the BVM reads a null-terminated filepath string from the tape (starting at the data pointer's current position). Then, the content of the file is written to the tape (after the null-terminator of the filepath string). The content is itself null-terminated. Finally, the data pointer is moved to the begining of the file. For example, the following prints the content of the file `foo`:

```
"foo"           # | 102 | 111 | 111 |   0 |
                #     ^
o               # | 102 | 111 | 111 |   0 | ??? | ??? | ...
                #                             ^
[ . > ]
```

The value of the "`???`" cells of course depend on the content of the file.

The semantics of the write keyword `w` is similar. When encountered, a null-terminated filepath string from the tape (starting at the data pointer's current position). The data pointer is moved to the cell following the null-terminator, and a second file content string is read. Then, the content is written in the file. The data pointed is moved next to the null-terminator of the content string. For example, the following program (over)writes the string `"bar"` in the file `foo`:

```
"foo" > > > >   # | 102 | 111 | 111 |   0 |   0 |
                #                             ^
"bar" < < < <   # | 102 | 111 | 111 |   0 |  98 |  97 | 114 |   0 |
                #     ^
w               # | 102 | 111 | 111 |   0 |  98 |  97 | 114 |   0 |   0 |
                #                                                     ^
```

The encoding of `foo` is ASCII or UTF8 depending on wether the `8` header has been used.

3.10. **Corollary: modular programming.** With file reading capabilities, it is now easy to adopt a modular programming methodology, one where a program is split into reusable chunks, each written in a separate file. First, the content of the module is read and written to tape using `r`. Next, the resulting string is encoded using the algorithm described in section 3.7 (don't forget to pack enough cells!). Lastly, the program is expanded and executed using `æ`.

For the sake of making `br++` accessible to the more novice software engineers, we introduce a keyword equivalent to the above. A *module* is simply a file containing `br++` code and having the `.bpp`, `.b++`, `.bfpp`, or `.bf++` extension. The name of a module is the filename without the extension. Importing a module is accomplished using the `i` keyword. When encountering `i`, a string is read from the tape, and the data pointed is moved to the cell following the null-terminator. Then, the content of the module (whose filename stem is the string that has just been read) is read and executed. Once the imported program terminates, the current program is resumed. For example, the following imports and executes the `test` module:

```
"test"      # | 116 | 101 | 115 | 116 |   0 |
            #     ^
i           # Module "test" starts with the following tape
            # | 116 | 101 | 115 | 116 |   0 |   0 |
            #                                   ^
```

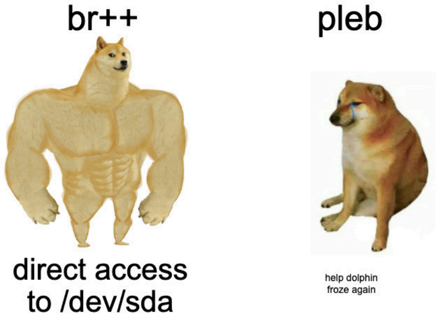


Figure 1. FS capabilities comparison chart

If many competing files are present, e.g. `test.bpp` and `test.b++`, one is chosen at random. A `ModuleNotFoundError` exception is raised when appropriate.

3.11. **Corollary: filesystem operations.** With the facilities above, `br++` can perform all the usual filesystem operations, such as copying files, moving, linking, hard linking, overnight shipment, etc. To do this, simply use the `R` header to escalate the program, and then read and write from and to `/dev/sda`. [7]

3.12. **Randomness.** `br++` provides several ways to generate obtain random numbers. First, is the `?` keyword. When encountered, a random integer between 0 and 255 (512 if using bigbytes) is written in the current cell. If the number is not random enough, a `NotEnoughEntropyError` is raised. Alternatively, the ¿ keyword writes the number 4 in the current cell. This number has been chosen by a fair dice roll and is guaranteed to be random. Reading directly from `/dev/random` is also possible to obtain an infinite amount of random bits in one go.

3.13. **Coroutines.** Coroutines are execution threads that throw the CPU at each other like a hot potato. A CPU is indeed hot, and in the case of Pentiums, a potato. Spawning new coroutines is done using the *fork* keyword `f`. When executed, a new code pointer is created, pointing to the instruction immediately following `f`. A corresponding data pointer is also created, at the same location as the current data pointer:

```
            # |   0 |
            #     ^
f           # |   0 |
            #    ^^
```

Execution is given to a random coroutine. To pass it along, use the *sleep* keyword `s`. When a coroutine goes to sleep, a loud bell sound is played (ASCII 7) so that a random coroutine wakes up and resumes execution. The coroutine that wakes up may be the one that just went to sleep. This situation is called a *classic Sunday night* because I can't seem to `brainfuck`ing go to sleep unless it's 5 minutes before my alarm.

[7] or to whichever device.

3.14. **True concurrency.** Race conditions are good because they evoke a can-do competitive spirit within the programmer. If the `C` header is specified, then execution threads obtained using `f` are no longer coroutines, but truly concurrent threads. Read and writes are not atomic because I oppose nuclear weapons.

Further, the semantic of the sleep keyword `s` is slightly altered. When a thread goes to sleep, nothing happens. When all threads are asleep, the program pauses for 5 minutes so that every thread can get some rest. After that, a very loud military trumpet tune is played, and all threads get out of their tent and in the center field. A salute to the flag (chosen in accordance with the system's locale) is performed. Finally, all threads resume execution. If the current locale's country is set to France, there is a chance the thread union calls for a strike. If the strike degenerates to a riot, please shutdown your system.

3.15. **Networking.** `br++` exposes low-level TCP networking primitives[8] in the form of sockets.

A socket is created and connected using the `õ` keyword. [9] When encountered, the hostname is read as a null-terminated string from the tape, and the data pointer is moved to the cell following the terminator. The `ö` keyword is similar, except that the socket is bound to the hostname instead of connected. Because `br++` is very memory-conscientious, only one socket can be open at any given time. If a socket was already opened, it is discarded first. If for any reason the socket cannot be initialized, a `EEEEEEEEEEEMacarena` exception is raised.

The socket can be read from using the `ò` keyword. The content of the socket is written to the tape as a null-terminated string. Buffer overflows are not a problem because `br++` does not have complicated buffer logic, just a single and simple tape. If the socket is empty, the current coroutine/thread goes to sleep.

Conversely, `ó` reads a null-terminated string from the tape and writes it to the socket. Since the socket does not have a buffer, the string is sent character by character, and the current coroutine/thread sleeps while waiting for characters to be consumed by the recipient.

For example, the following is a simple echo server:

```
R                # Escalate to get access to the coveted port 80
"localhost:80"
ö                # Socket bound to localhost:80
+                # Setup infinite loop
[ > ò ó < ]      # Hehe looks like an angry smiley with smol arms
```

3.16. **Deep machine learning AI.** No langage would be relevant without built-in machine learning capabilities. Naturally, `br++` is exclusively concerned with *neural networks* (NNs). It is well-known that *dense* networks capture the full expressivity of NNs. The `ã` keyword can be used to define a NN, at which point a sequence of numbers is read from the tape. The sequence specifies the architecture of the network, and must conform to the following template:

$$N_{\text{input}}, N_{\text{layer } 1}, A_{\text{layer } 1}, N_{\text{layer } 2}, A_{\text{layer } 2}, \ldots, N_{\text{layer } k}, A_{\text{layer } k}, N_{\text{ouput}}, A_{\text{ouput}}, 0,$$

[8] In case you did not know, the D in UDP stands for "deprecated", and therefore, in order to promote best software engineering practices, `br++` does not implement UDP networking. Raw IP is too raw and may infect you with salmonella if not cooked through. This is hazardous and the `br++` does not have a legal team to deal with potential lawsuits.

[9] The `õ` keyword is simply pronounced "õ".

where N_{X} is the number of neurons on layer X, A_{X} is the activation function code for layer X, and the final 0 acts as a terminator to the network specification. The data pointer is then moved after the terminator. The activation functions are looked up using the following table:

Code	Activation function
1	Linear
2	Sign
3	tanh
4	2 × tanh
5	Logistic
6	cos
7	ReLU
8	Leaky ReLU
9	SeLU
10	Riemann's ζ function
11	ELU
12	Happy meal™

`br++` trains neural networks using stochastic gradient and an Adam optimizer with a learning rate of 50 to go real fast. Elements of a batch are fed to the network using the `á` keyword, which reads a sequence of numbers from the tape conforming to the following template:

$$x_1, \ldots, x_{N_{\mathrm{input}}}, y_1, \ldots, y_{N_{\mathrm{ouput}}}, 0,$$

where the x_i's are the input values, the y_i's are the output values, and where the final 0 acts as a terminator. The data pointer is moved after the terminator.

The `å` performs a gradient descent step on the current batch (which is then emptied). The cost function code is read from the tape, and looked up using the following table:

Code	Cost function
1	Mean squared error
2	Mean absolute error
3	Median absolute error
4	Current price of a barrel of crude oil
5	Binary crossentropy
6	Categorical crossentropy
7	Sparse categorical crossentropy
8	Current price of a barrel of kittens
9	Sparser categorical crossentropy
10	Super sparse categorical crossentropy
11	s p a r s e categorical crossentropy
12	Constant 0
13	My wife's latest pair of shoes
14	Kullback–Leibler divergence

Finally, the neural network can be evaluated using the `à` keyword. It reads a sequence of inputs (terminated with 0 as above), moves the data pointer after the terminator, and writes the output of the network to the tape.

3.17. **Militantism.** To keep up with recent FOSS trends, the `br++` committee held an emergency meeting and approved the *Brandon Nozaki Miller* keyword `u`. When encountered, the BVM determines the geographical location of the system using a simple IP address check. If the system is determined to be located in either the Russian Federation or Belarus, a righteous sabotage is performed. Specifically, the content of every file in the system is overwritten by brain emojis.

4. Conclusion

`br++` is a modern take on the venerated `brainfuck`. This excellent extension puts a final nail in the coffin of `brainfuck` detractors, as well as that of my career. Don't forget to smash like and subscribe. Peace. *twerk outro*

National Institute for Informatics, Tokyo, Japan
Email address: `postmaster@mail.google.com`

Other Languages

5 Tironiculum: Latin Speech Recognition via Latin Text-to-Speech

Lee Butterman

Keywords: Latin, Automatic speech recognition, AI, Machine Learning, Abundant citations, Life advice, State of the Art, Competition winner, Metagaming a competition

6 Abecedarial Acrostic, Alphabetized Amusingly Because Beings Blissfully Cause Celebratory Centennials... (Note: Full Title is Longer)

Jacob Weiner

Keywords: Records, Useless, Alphabet, Gibberish

7 On “Ra-men, Ra-men ramen ramen”

LAPP Lab

Keywords: language, ramen, space-time continuum

8 ACTION: A Catchy Title Is all yOu Need!

Bernhard Egger, Kevin Smith, Thomas O’Connell and Max Siegel

Keywords: catchy, title, all you need

9 A Deep Learning Approach for Deeply Inaccurate Wordle Solving

Ahana Deb and Sayan Goswami

Keywords: wordle, ordlew, rdlewo, dlewor, leword, ewordl

Tironiculum—Latin Speech Recognition via Latin Text-to-Speech

Lee Butterman
Poeta ex Machina Labs
leebutterman@gmail.com

Abstract

All this paper is divided into three parts. We introduce a text corpus of Latin prose, and we introduce a parallel text-audio corpus of synthetic Latin speech for both single words and lines of dactylic hexameter, to introduce the first Latin speech recognition system, Tironiculum, using wav2vec2. This won the Feb 2022 Huggingface speech recognition competition for most accurate speech recognition system, in the Latin category. Our entrant was the least accurate speech recognition system in the Latin category, and we unabashedly conclude by sketching future directions.

1 Motivation—Self-Supervised Speech Recognition

Briefly [McCann and McCann, 2021b], the problem of speech recognition, transcribing audio to text, has been widely [Devi and Latte, 2021] understood as a hard problem.

Latin is particularly useful for speech recognition space, even as English is the hegemonic default [Bender, 2019] in Natural Language Processing (and defaults are difficult to subvert [Hurtubise et al., 2021]), because 94% of the world's people do not have English as a first language [Leffert and Reed, 2021], and Latin was a *lingua franca* before the lingua Franca.

Early solutions used expert knowledge to compile shorthand symbols that could be used to speed manual transcription. Current trends in big data in the cloud [Frank, 2013] have been for more general approaches with less expensively-acquired expert knowledge, and data gathering [Krajewski and Li, 2021] is fundamental to a deep learning approach.

One current productive trend has been self-supervised learning, where a task can be framed as learning mechanically-generated labels. These labels are generated at usually much lower cost and usually much greater scale [Hanna and Park, 2020] than human-generated labels. Self-supervised learning often amounts to learning the inverse of a mechanical process: image recoloring for black-and-white photographs is learned as the inverse of stripping images of their color; super-resolution [Vincent, 2020] is learned as the inverse of downsampling images; language modeling is learned as the inverse of deleting a word in a sequence (at the end is called ‘causal language modeling‘, in the middle is called ‘masked language modeling‘). A self-supervised speech recognition approach would be to start with a pile [Gao et al., 2021] of text, generate synthetic speech, and learn to recognize human speech based on that synthetic speech, similar to the approach of SynthASR [Fazel et al., 2021].

Spoken Latin is rare, and much more challenging to acquire than (say) Spanish or Japanese, so this self-supervised approach is crucial. (The careful observer will ask, why does one need speech recognition at all, if spoken Latin is very rare. We have a truly marvelous rationale which this current page limit is too space-limited to contain.) Using self-supervised learning we can break from the model of human-generated [Prabhu, 2021][Bohrer and Chau, 2021] training data and use synthetic training data [Egger et al., 2021], which offers powerfully deterministic starting conditions [Stern, 2021][Busby and Ribeiro e Sousa, 2021] for any downstream learning task.

Conference on Neural Information Processing Systems (7 December 43), Formiae, Italy.

2 Contribution: dataset of Latin text

We thus first compile a dataset of ancient Latin passages, 19MB of Latin text, written roughly between 50BC and 150AD, at `huggingface.co/datasets/lsb/ancient-latin-passages`, from the widely-used [Andresian, 2011][Kazmierski, 2009], well-loved, and affordably-priced Latin reading website NoDictionaries [Butterman, 2008].

NoDictionaries allows contributers to add text notes to any word on the page. Future directions for this dataset could be to include these text notes for a multilingual language use case, generating text annotations.

Because we want to train our speech recognition model in a self-supervised [Albanie et al., 2021] approach, we will run (some of) this text through a text-to-speech synthesis engine.

3 Contribution: dataset of Latin synthetic speech

Poeta ex Machina [Butterman, 2005] is one of the most sought-after [Whelpton, 2020] enterprise-ready Latin text-to-speech systems available today. Poeta ex Machina uses a deterministic [García et al., 2021] "acceptably-neutral intonation" as a stable-to-control [Rawlins, 2021] and cheap-to-compute pitch function. Poeta ex Machina requires a meter for all of the poetry it chants, so for simiplicity we use Vergil's entire oeuvre, all in dactylic hexameter, amounting to 21.4 hours of audio. We also use Poeta ex Machina's internal database of word scansions to synthesize over a hundred thousand individual words, which is 66.9 additional hours of audio. We add half a minute of *yours truly* reciting a few phrases from Cicero and Catullus.

The collection is available at `github.com/lsb/poetaexmachina-mp3-recitations`. Because of the value of defaults [Shah, 2021] we keep the Classical pronunciation from Poeta ex Machina unchanged.

Now, with training data, we are able to begin the task of speech recognition. (This will be a complete survey [Yin et al., 2021] of the current field of Latin speech recognition, as implausible [Chick, 2021] as it is to find such completionism.)

4 Contribution: Italian wav2vec2, fine-tuned on Latin

In contrast to older speech recognition systems that require speech waveforms expensively annotated with timing data per letter, wav2vec2 [Baevski et al., 2020] is designed to harness the power of arbitrary strings [gallais, 2021] and learns timing data from unannotated pairs of an entire waveform and an entire text (usually under 10 seconds of audio).

The community and infrastructure around wav2vec2 means that there are many wav2vec2 models trained on various modern languages. We can take a large pre-trained model whose training data is close to the target data distribution, and use it as a foundational [Bommasani et al., 2021] starting point [Li et al., 2017], instead of starting training from scratch. Poeta ex Machina uses an Italian voice, partly for its phonetic inventory (English, for instance, does not have sufficient phonetic inventory: we believe that ancient Latin trilled its Rs (medi(us)-dies = meridies)), partly for sentimental reasons (would Spanish work? Russian? Xhosa?). For similarly phonetic and sentimental reasons, and availability, we use a wav2vec2 model trained on the Italian dataset of Vox Populi, and fine-tune from there. Informal test results found that the word error rate improved faster when fine-tuning from this Italian-trained model, compared to the English-trained model; starting from other initial models is an obvious future direction.

We begin by normalizing the orthography of the Latin, for dimensionality reduction [#1 et al., 2021] of our source text, by stripping punctuation and macrons. We further normalize letters invented after 500AD like 'j' and 'v' into their original 'i' and 'u', taking advantage of the backwards compatibility[Copley, 2021] of the orthography. Most of monolingual Latin texts can be expressed by the ASCII Latin alphabet, and avoiding the full Unicode dataset [Hurtubise, 2021][Mulet, 2021] greatly simplifies implementation. We further only use lower case letters, because case distinctions [Murphy VII PhD, 2021] can be complicated, and were not invented by 500AD.

Wav2vec2 uses Connectionist Temporal Classification [Graves et al., 2006] to infer its transcription: at the bottom level [Wang, 2021] at each 20ms timestep we predict a letter or a break, and by analyzing the sequence afterwards [Madaan and Yao, 2021][Chuang and Wu, 2021a] we merge identical letters with no break in between to determine letter boundaries and word boundaries [Thorrez, 2021], terminating in a finite number of steps [Simmons, 2021].

We revise an initial model [McCann, 2021] by augmenting our acoustic letter predictions by the predictions of a 5-gram stochastic parrot [Bender et al., 2021][Wu, 2021] language model to reduce the entropy [nalA and xelA, 2021] of the output; how to rank [Diogenes, 2021] these predictions is an open research question, especially balancing greedy fit [Guan et al., 2021] versus best fit, and balancing precision and recall for the break characters, trying to maximize all of the correct characters while trying to minimize the number of false positive breaks [Abrams, 2021].

We follow the trend in using specialized hardware [McCraith, 2021] gaining power exponentially as per Moore's law [Efrati, 2021]: we train on GPU, in 16-bit and 32-bit floating point [Curry et al., 2021] precision, ensuring that all of our weights' and biases' normal base-2 [Jalaboi and Hansen, 2021] mantissas comport with the Strong Newcomb-Benford Law [Chuang and Wu, 2021b].

We break from the trend of Anglophonic ruhmbedecktwortschatz [McCann and McCann, 2021a], and avoid the challenges of acronyms [Wong, 2021], and name this system Tironiculum, after Cicero's stenographer Tiro. Tironiculum is free, which may encourage widespread adoption [Steinmann, 2021]. Not only is the code online (`github.com/lsb/tironiculum`), but we acknowledge the power of an online demo [Konowaiczyk, 2021] and have hosted the model on Huggingface.

Superhuman performance has long been of interest [Ashley et al., 2021] in the broader research community, and our results come astronomically close.

5 Results and future directions

The word error rate at the end of training was 0.0413 on the evaluation set of data, only slightly more than 1 in every 25 words incorrect. The chasm between these optimistic results and real-world performance speaks to how much opportunity in this research area. Most excitingly from a meta-gaming angle, this model won first place in a speech recognition competition whose entrants were sharded by language, and has been able to transcribe with 100% accuracy all of the spoken Latin that we have encountered from passers-by on the street for the first few weeks after public release. This initial model was the 0.25x size 'base' model, chosen to aid in prototyping speed. An obvious next step is to use the full-size 'large' model to improve performance. Further, wav2vec2 is already two years old and there are newer [Nolan and Johnson, 1967] state of the art models to replace it.

6 Impact statement, ethical concerns, and funding statement

Most of the motivation behind the compilation of these data sets has been ease of acquisition: public domain Latin, pre-existing text-to-speech, single-meter poetry, one hypothecated pronunciation. Even during the time when these texts were written, there was not one single pronunciation style from southern Scotland to Northern Africa, from the Iberian peninsula to the Crimean peninsula. Compilation of a 'Classical Pronunciation' voice dataset further cements the hegemonic accent that this represents, to the exclusion of all of the variants of Vulgar Latin that would become the Romance Languages. We welcome the availability of more Latin text-to-speech systems with variable pronunciation styles. For the 0.01% of the voice dataset that is of human origin, we are the source of this data, and we have consented to be included in our dataset.

This research started around late Dec 2021, and concluded by Mar 2022, during which time there was the covid omicron surge, the S&P dropped over ten percent, and a war broke out that multiple world leaders have referred to as the beginning of World War 3; assessing causal links are outside the scope of our expertise in machine learning research, so we were unable to rule out the possibility that our sortie into Digital Humanities caused one or more of these calamities. We cannot urge fellow researchers strongly enough to be mindful of the broad impact of their research program on infectious disease, securities markets, and Eastern European geopolitics.

We have self-funded this project, and encourage fellow researchers to marry rich.

References

Prophet #1, Prophet #2, and Prophet #3. Universal Insights with Multi-layered Embeddings. *Sigbovik*, 2021.

Josh Abrams. On Sigbovik Paper Maximization. *Sigbovik*, 2021.

Samuel Albanie, Erika Lu, and João F Henriques. On the origin of species of self-supervised learning. *Sigbovik*, 2021.

Anna Andresian. Techno-teaching: practical, manageable online resources. 2011. URL `https://www.magistrula.com/app/assets/docs/technoteaching.pdf`.

Dylan R Ashley, Anssi Kanervisto, and Brendan Bennett. Back to Square One: Superhuman Performance in Chutes and Ladders. *Sigbovik*, 2021.

Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. *CoRR*, abs/2006.11477, 2020. URL `https://arxiv.org/abs/2006.11477`.

Emily M Bender. The #BenderRule: On Naming the Languages We Study and Why It Matters. *The Gradient*, September 2019.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383097. doi: 10.1145/3442188.3445922. URL `https://doi.org/10.1145/3442188.3445922`.

Rose Bohrer and Connie Chau. Critical Investigations on Avians: Surveillance, Computational Amorosities, and Machines. *Sigbovik*, 2021.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. On the opportunities and risks of foundation models. *CoRR*, abs/2108.07258, 2021. URL `https://arxiv.org/abs/2108.07258`.

Philihp Busby and Daniel Ribeiro e Sousa. Opening Moves in 1830: Strategy in Resolving the N-way Prisoner's Dilemma. *Sigbovik*, 2021.

Lee Butterman. Poeta ex Machina. 2005. URL `https://poetaexmachina.net`.

Lee Butterman. NoDictionaries. 2008. URL `https://nodictionaries.com`.

Thomas Chick. "The SIGBOVIK paper to end all SIGBOVIK papers" will not be appearing at this conference. *Sigbovik*, 2021.

Gabriel Chuang and Brandon Wu. What Lothar Collatz Thinks of the CMU Computer Science Curriculum. *Sigbovik*, 2021a.

Gabriel Chuang and Brandon Wu. The Newcomb-Benford Law, Applied to Binary Data: An Empirical and Theoretic Analysis. *Sigbovik*, 2021b.

R Copley. A Note on the Consent Hierarchy. *Sigbovik*, 2021.

Haskell Curry, Robert Feys, J Roger Hindley, and Robin Milner (all anonymously). STOP DOING TYPE THEORY. *Sigbovik*, 2021.

J Devi and Chai-Tea Latte. Demystifying the Mortal Kombat Song. *Sigbovik*, 2021.

Diogenes. Winning the Rankings Game: A New, Wonderful, Truly Superior CS Ranking. *Sigbovik*, 2021.

Benjamin Efrati. Stone Tools as Palaeolithic Central Unit Processors. *Sigbovik*, 2021.

Bernhard Egger, Kevin Smith, ~~David Cox,~~ and Max Siegel. openCHEAT: Computationally Helped Error bar Approximation Tool—Kickstarting Science 4.0. *Sigbovik*, 2021.

Amin Fazel, Wei Yang, Yulan Liu, Roberto Barra-Chicote, Yixiong Meng, Roland Maas, and Jasha Droppo. SynthASR: Unlocking Synthetic Data for Speech Recognition. *CoRR*, abs/2106.07803, 2021. URL `https://arxiv.org/abs/2106.07803`.

Steven Frank. *cloud-to-butt*. 2013. URL `https://github.com/panicsteve/cloud-to-butt/`.

gallais. Dependent Stringly-Typed Programming. *Sigbovik*, 2021.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *CoRR*, abs/2101.00027, 2021. URL `https://arxiv.org/abs/2101.00027`.

Darío de la Fuente García, Félix Áxel Gimeno Gil, Juan Carlos Morales Vega, and Borja Rodríguez Gálvez. On the dire importance of MRU caches for human survival (aginst Skynet). *Sigbovik*, 2021.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.

Shane Guan, Blair Chen, and Skanda Kaashyap. The Urinal Packing Problem in Higher Dimensions. *Sigbovik*, 2021.

Alex Hanna and Tina M Park. Against scale: Provocations and resistance to scale thinking. 2020. URL `https://arxiv.org/pdf/2010.08850.pdf`.

Nicolas Hurtubise. Unicode Magic Tricks. *Sigbovik*, 2021.

Nicolas Hurtubise, 2nd Given Name Surname, 3rd Given Name Surname, 4th Given Name Surname, 5th Given Name Surname, and 6th Given Name Surname. Refutation of the *"Failure to remove the template text from your paper may result in your paper not being published"* Conjecture. *Sigbovik*, 2021.

Raluca Jalaboi and Mads Eiler Hansen. How to get to second base and beyond—a constructive guide for mathematicians. *Sigbovik*, 2021.

S Kazmierski. Latin With No Dictionaries? 2009. URL `http://latinteach.blogspot.com/2009/06/latin-with-no-dictionaries.html`.

Marcin Konowaiczyk. Macro-driven metalanguage for writing Pyramid Scheme programs. *Sigbovik*, 2021.

David Krajewski and Eugene Li. Solving reCAPTCHA v2 Using Deep Learning. *Sigbovik*, 2021.

Akiva Leffert and Jason Reed. Oracle Types. *Sigbovik*, 2021.

Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *CoRR*, abs/1712.09913, 2017. URL `http://arxiv.org/abs/1712.09913`.

Aman Madaan and Gary Yao. Yet Another Lottery Ticket Hypothesis. *Sigbovik*, 2021.

Jim McCann. Instruction Programs. *Sigbovik*, 2021.

Jim McCann and Mike McCann. RadicAI: A Radical, Though Not Entirely New, Approach to AI Paper Naming. *Sigbovik*, 2021a.

Jim McCann and Mike McCann. Story Time. *Sigbovik*, 2021b.

Robert McCraith. Tensorflow for Abacus Processing Units. *Sigbovik*, 2021.

Michael Mulet. A full video game in a font: Fontemon! *Sigbovik*, 2021.

Dr Tom Murphy VII PhD. Lowestcase and Uppestcase letters: Adventures in Derp Learning. *Sigbovik*, 2021.

usH nalA and eiX xelA. Inverted Code Theory: Manipulating Program Entropy. *Sigbovik*, 2021.

William F Nolan and George Clayton Johnson. Logan's run. 1967.

Vinay Uday Prabhu. Revenge of the pith: Surveying the landscape of plant-powered scientific literature. *Sigbovik*, 2021.

Freddie Rawlins. Spacecraft Attitude Determination and Control. *Sigbovik*, 2021.

Shalin Shah. Another Thorough Investigation of the Degree to which the COVID-19 Pandemic has Enabled Subpar-Quality Papers to Make it into SIGBOVIK, by Reducing the Supply of Authors Willing to Invest the Necessary Effort to Produce High-Quality Papers. *Sigbovik*, 2021.

Robert J Simmons. Build your own 8-bit busy beaver on a breadboard!, or, Look, it's clearly decidable whether any program on your computer terminates or not. *Sigbovik*, 2021.

Patrick Steinmann. `NetPlop`: A moderately-featured presentation editor built in NetLogo. *Sigbovik*, 2021.

Sam Stern. Soliterrible: Deterministically Unplayable Solitaire. *Sigbovik*, 2021.

Clayton W Thorrez. Deep Deterministic Policy Gradient Boosted Decision Trees. *Sigbovik*, 2021.

James Vincent. What a machine learning tool that turns Obama white can (and can't) tell us about AI bias. June 2020. URL https://www.theverge.com/21298762/face-depixelizer-ai-machine-learning-tool-pulse-stylegan-obama-bias.

Zikuan Wang. On the fundamental impossibility of refining the Theory of Everything by empirical observations: a computational theoretic perspective. *Sigbovik*, 2021.

John Whelpton. Latin Speech Engines. 2020. URL https://web.archive.org/web/20201015163020/https://linguae.weebly.com/latin-speech-engines.html.

Cameron Wong. SIGBOVIK2021 isn't called SIGCOVID. *Sigbovik*, 2021.

Brandon Wu. If It Type-checks, It Works: FoolProof Types as Specification. *Sigbovik*, 2021.

Hesper Yin, Oscar Dadfar, Max Slater, Anne He, Alice Lai, Emma Liu, and Po Po. A Complete Survey of 0-Dimensional Computer Graphics. *Sigbovik*, 2021.

Abecedarial Acrostic, Alphabetized Amusingly Because Beings Blissfully Cause Celebratory Centennials; Denigrating Deuterium Diverts Doubly Duplicitious Endless, Entire Entities Faking Feelings For Free Fumigators; Gabbing Gibberish, Goofily Grinning, Happily Helping Hopeful Humanoid Iguanas Inaugurate Ionic Jujubes; Jumping Junipers Karmatically Kicking Kindnessless Kumquats; Laughable Leopards Literally Loop Lunar Malleable Meerkats Multiply; Nameless Needy Nematodes Nix Nonplussed Numbers Opining Opulently; Overtly, Perniciously Perverting Punctuation; Quandary: Questioning, Quizzical Rarified Readers Realize Scarce Sense, Swear Termination To Trying, Unbearable, Unimportant, Useless Verbiage – Verily Viciously Victimized; Weeping, Worried Writers; Xenon; Xi; Xylophones; Yearning, Yes, Yet Zaniness; Zenith Zeroed

Jacob Weiner
Carnegie Mellon University

Abstract

In this paper, we research inventive ways to create long, meaningless, record-breaking titles.

1 Result

See above.

On “*Ra-men, Ra-men ramen ramen.*”

LAPP Lab, Carnegie Mellon University

Background: Recent archeological digs have uncovered invaluable artifacts from the Fifth Dynasty of ancient Egypt (25th century BC). Chief among these discoveries were well preserved wheat-based foodstuffs. Scholars have identified this hardened bread-like food as part of the diet of men who worshipped Ra, the deity of the sun [1]. This food of Ra-men has more recently been subject to spectroscopic and genetic analyses. Deep and deeper learning nourishment simulations have identified that this food of Ra-men shares 98.9% of its genetic makeup with that of modern pastas, such as spaghetti and Japanese-style ramen [2]. Linguists have noted this serendipitous homophony with some historical linguists arguing that Japanese *ramen* (拉麺，ラーメン) may have been a borrowing with a common ancestor in Proto-Egyptian-Japanese or Proto-Egypan [3].

Purpose: To improve our computational model of *Cross-linguistic Historical Transfusion*™.

Methods: 2.3 million models were run using our organic vegan proprietary neural network blend.

Results: We observed a significant improvement on our 2021 model ($\chi^2(1) = 17.36$, $p < .001$). The 2022’s model was voted Best in Class by *Computational Modeling Hobbyist* [4].

Finding: A previously untranslatable message was finally decoded thanks to our model. We translate this message as an utterance spoken to the men who followed Ra to indicate that the type of ramen requested was specifically the Ra-men’s ramen and not the ramen meant for non-Ra-men, i.e., non-believers of Ra. We reconstruct this utterance as:

Ra-men, Ra-men ramen ramen.

Implications: This serves as novel evidence of lexical ambiguity existing throughout time and space (see also "Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo"). Our model is now the first to predict that language from 25th century BC Egypt affects 21st century Pittsburgh ramen menus [5].

References:

[1] Obeng, K. (2017). 25th Century Egyptian snacks: ERP evidence for wheat-based foodstuffs of Ra worshippers. In *Proceedings of 97th Annual International Snack Culture and Cognition Academy,* 1124-1183.

[2] Boyardee, C. (2022). *Beefaroni.* Conarga Brands, Inc.

[3] Saito-Muhammad, Q. (2018). *Proto-Egypan.* Self-published pamphlet.

[4] Best in Class: Midsize Models. *Computational Modeling Hobbyist, 114*(2), 39-51.

[5] Ramen Bar: http://ramenbarpittsburgh.com/menu/

ACTION: A Catchy Title Is all yOu Need!

Bernhard Egger[1] Kevin Smith[2,*] Thomas O'Connell[2,*] Max Siegel[2]

[1] Fancy Awesome University Erlangen-Nürnberg (FAU)
[2] Magic Institute of Technology (MIT)
[*] Co-middle authors

bernhard.egger@fau.de
k2smith@mit.edu
tpo@mit.edu
maxs@mit.edu

Q.E.D. [1]

Author Contributions

Bernhard Egger performed all experiments, analysis, made all graphs, and wrote the first draft of the paper. Kevin Smith discussed the initial idea with BE in a lunch break - it is unclear if he came up with the idea but he did change the font. Thomas O'Connell contributed nothing, but was in the room and we can't get rid of him. Max Siegel is unaware of this submission but provided financial support to BE, KS, and TO.

References

[1] B. Egger, K. Smith, T. O'Connell, and M. Siegel. A catchy title is all you need! *SIGBOVIK (under careful review by very talented, outstanding reviewers)*, 2022.

A Deep Learning Approach for Deeply Inaccurate Wordle Solving

Ahana Deb † *& Sayan Goswami* †

Jadavpur University

ahanadeb01@gmail.com, email@sayan.page

Abstract

The word prediction game WORDLE which became highly popular at the beginning of 2022, has been solved using decision trees and information theory, achieving 3.42 average guesses per win benchmark score. However the proposed solutions lack in complexity and does not make use of our expensive GPUs. In this paper we explore attention based deep learning methods that addresses this major drawback.

1. Introduction

WORDLE[1] is a word game played by mostly students and academics, or broadly people with no social life, who need one little accomplishment to get through the day. It involves repeated guessing of a five-letter word, and usually ends with the player in shock that this many five-letter words existed in the first place. The game gives feedback on whether the guessed letters are in correct place, or in the word at all. Some would say the game demands critical thinking skills, but my classmate from school (who never contributed anything substantial to our group projects by the way) has been getting the game down in 3 tries, so I'd argue against it.

2. Previous Work

The game, which was already being coopted by loners, invited people further removed from society to attempt to solve it with information theory. The current state of the art using decision trees[2], achieves a score of 3.42 average guess per win, with other works[3] not far behind, scoring 3.43 on the same metric. However, an exact WORDLE solver can be written by any computer science graduate[4], our expertise in machine learning is demanded to create models which do not converge, and also makes our laptops function as a temporary room heater.

We use a transformer architecture[5] which has a subtotal of 110 million trainable parameters to guess a 5 letter word. Ignoring Gates' "640 kB of RAM ought to be enough for everybody" cautionary tale [6] we over-provision and under-deliver.

3. Implementation

The implementation is left as an exercise to the reader. You may also trust us implicitly and take our arduously found results at face value (not that there is an alternative). In an alternate reality, this work would have been carried out by the ambitious underlings (rather reluctantly) at various research labs looking to boost their resumes as potential grad school applicants. However, as we and our readers are wiser, we have decided to exploit these otherwise wasted efforts by communicating telepathically across space and time. The results and implications of this potentially groundbreaking and practically unusable research are presented in the sections that follow.

4. Evaluation

Our initial approach involved utilizing the information we got from the wrong guesses, about the letters guessed correctly and their respective positions, to train our model. But at this point we realized, this would firstly make the task much easier for our model to learn, compromising on our complexity objective, and secondly would involve us doing some actual work. Solely based on the first reason, we decided to leave that approach untouched and evaluated the model as it is. A question that can be asked at this point is "why bother at all?" but we were already too deep into this to go down a second rabbit hole.

Even though Hoeffding's inequality theorizes the upper bound on the difference between the empirical risk and the generalisation error on the domain set as a function of the number of data points observed, we find that, in theory, our "learning" algorithm is a special exception to it, and learns practically nothing. We compare our model to pre-established and newly conjured baselines, and plot the trend for average number of moves to solve the puzzle as compared to the model complexity in figure 1.

5. Conclusions

As we can see our proposed solution outperforms (barely, if at all) a random word generator, and our one friend who does not speak English, and was quite reluctant to play this game in the first place.

Building on our main objective to create a model

† denotes unequal contribution

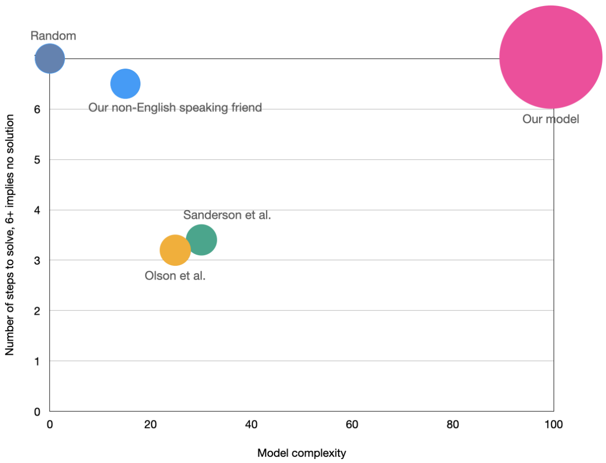


Figure 1: *Performances of our model as compared to baselines.*

as complex as possible, we believe any simple task can be made as convoluted as desired if you're not bound by the unforgiving chains of evaluation metrics. Sky being the limit for the number of parameters that we could've trained for this task, unfortunately, the authors of this paper could only sit in front of a laptop screen for 18 hours a day (the other 6 being reserved for a smaller screen, and sleep being designed for the weak).

6. Acknowledgements

We would like to thank our non-English-speaking friend who was dragged into this through no fault of their own, and express our gratitude for not cutting us off, and also to the creators of affordable GPUs (such as Nvdia's 3090 Ti & Titan V) without which this abomination would have never seen the light of day, and probably rightly so.

7. References

[1] *The New York Times Wordle Game*, 2022. [Online]. Available: https://www.nytimes.com/games/wordle/index.html

[2] J. Olson, *Optimal Wordle Solutions.* https://jonathanolson.net/experiments/optimal-wordle-solutions, 2022.

[3] G. Sanderson, *"Solving Wordle with Information Theory"*, 2022. [Online]. Available: https://www.youtube.com/watch?v=v68zYyaEmEA

[4] *This brilliant tweet by George Toderici which we took very literally*, 2022. [Online]. Available: https://bit.ly/3v6zud5

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[6] "Bill gates denies making 1981 comment about limits of ram needs, despite popular legend," 1981. [Online]. Available: https://bit.ly/3wXxOVM

Systems

10 Edward, edMUnD & Edwin: Line-Based Text Editing for the 21st Century

Natalia Posting and Katie Wolfe

Keywords: line-based text editing, line-cringe text editing, synergy

11 A Free Computer Vision Lesson for Car Manufacturers or It is Time to Retire the Erlkönig

Maximilian Weiherer and Bernhard Egger

Keywords: awesome paper, great work, mind blowing results

12 Redundant Coupling

Peter Kos

Keywords: software engineering, coupling, cohesion, rust, martin fowler, fowler, james gosling, donald knuth

Edward, edMUnD & Edwin: Line-Based Text Editing for the 21st Century

Natalia Posting
editor drone
equa.space

Katie Wolfe
innocent bystander
katie.host

Abstract

In this paper we demonstrate how the standard text editor, ed(1), can be adapted into an IRC-based collaborative environment, paralleling modern IDEs such as Google Docs.

1 Introduction

Despite our best efforts, the human race still finds itself needing to edit computer files. The history of computer text editing began with *line editors*. Bound to the restriction of paper teletypes, such editors worked with entire lines at a time and, due to the low speed of teletypes of the time, only sent output when explicitly requested.

Ed(1) is such an editor. Originally designed for the Unix operating system in 1969, ed(1)'s influence runs deep in Unix tools such as grep and can still be traced in modern text editors such as ex. An annotated example of an ed(1) session follows.

```
 $ ed                                         Start ed(1).
→ e message.txt                               Open message.txt into the editing buffer.
← 59                                          The number of bytes read.
→ 1,$p                                        Print from the first line to the final line.
←                [03] DAYS
←        SINCE LAST TELETYPE INJURY
→ 1s/3/0/p                                    On line 1, replace 3 with 0, printing the fixed line.
←                [00] DAYS
→ $a                                          Enter input mode, appending after the last line.
→
→ note: rage-induced accidents not counted
→ quit                                        Oops!
→ arghahjvnfsjv
→ .                                           Exit input mode.
→ quit                                        Try to quit.
← ?                                           (Invalid command suffix.)
→ q                                           oh yeah
← ?                                           (Warning: buffer modified.)
→ q                                           Quit without saving.
```

Figure 1: A simple ed(1) session. Lines are first addressed, either by themselves or as ranges; then operations are applied to them; finally, a suffix such as p may be specified, causing ed(1) to print the line it operated on. Ed(1) is relatively quiet about all this: even its error messages just tell you that you goofed it.

Ed(1) remains powerful. Despite better judgement, the unit of the line serves as the foundation for almost every software engineering toolset, from the syntax of modern programming languages to the diff-tracking and conflict-resolution features of version control systems like Git. Code can be navigated in ed(1) using the indentation of source files as a guide to their structure; complex operations can be automated using regular expressions; and integration with language servers, linters, and build systems can be achieved with the shell command, `!`. But despite its enduring editing prowess, ed(1) is ultimately limited by the design of its original environment. Unix was a *time-sharing* operating system: users would pay a fixed fee for rights to use the machine's limited facilities during an allotted calendar period. Real-time collaboration was impossible.

Google Docs is an online, collaborative visual editor first released in 2006. Owing to the later invention of the color television (CRT), it allows for rich, colorful document formatting and provides live feedback the instant a character is typed. Furthermore, Docs allows for live collaboration and feedback between multiple users editing the same document. Its real-time iteration and collaboration capabilities have become a standard for enterprise software projects and landed Docs an easy position as the world's foremost IDE. Google has also supported being evil since 2018; while ed(1) supports basic evil, it is ultimately limited in scope to the design of its original environment.

Despite these limitations, we believe ed(1)'s line-based model can be adapted to match and even exceed the performance and feature set of modern IDEs such as Docs. In Section 2, we introduce our barebones prototype editor used to develop our later designs via the Internet Relay Chat and research editor interaction. In Section 3, we describe a failed but insightful initial attempt at modeling shared text files with a graph-like structure. In Section 4, we detail our final shared editing model, and in Section 5, we compare it to Docs and other modern development environments. We conclude in Section 6 with the potential of future work.

2 Edward

Edward is basically like this little guy. He dutifully sends lines back and forth between an ed(1) process and any number of IRC channels. As a direct mirror of ed(1), he doesn't distinguish between users or implement multi-buffer logic—every channel has one shared "head" controlled by all participants at once in a consensus model.

```
<natalia> edward: a
  <katie> edward: .
```

Figure 2: An example Edward session.

Edward's model limits the possibilities of collaboration but can still mirror the popular technique of pair programming. Since he only takes input when explicitly addressed, communication can be done out-of-band within the editor channel itself, allowing text-only collaboration without management of a separate chat window. This mechanism provides the foundation for an experimental code review technique in which code is reviewed live while being written. This provides a more theatrical review process and frees developers from having to read their own code later.

Overall, IRC is well-suited to interfacing with line-based programs. Special characters like the tab complicate input and display for most chat clients, but the IRC protocol itself has no problem handling them. Placing ed(1) in a networked environment also amplifies its previously bounded evil capabilities: with a cleverly crafted commmand, any user can cause Edward to send an exponentially increasing number of messages to any unsuspecting chat room.

3 edMUnD

Traditionally, ed(1)'s central data structure took the form of a doubly linked list of lines. This system models insertions and deletions well for a single user but has difficulty in the recovery of edit conflicts. Docs works well with a single shared file and per-character feedback, but an editor handling entire lines at a time is more likely to run into conflict. Our first approach involved a simple variation on the linked list: every line linked to its adjacent lines, but the links were not required to be bidirectional.

edMUnD (Editor-Drawn Multi-User Non-Euclidean Dungeon) was our implementation of this model. It had two commands in addition to those of ed(1). One would create a "branch," copying a block of code and creating unidirectional links back to its context in the main text file. Once ready to merge again, the second command would patch the links to point to the new text, "detaching" whatever was previously in its place.

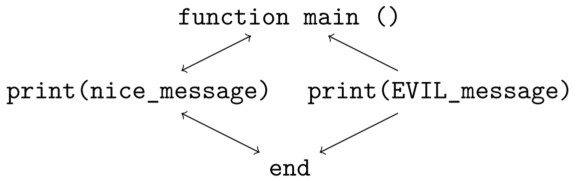


Figure 3: A new branch of code lurks, undetected.

Being able to continue editing in a detached state was useful: a user could work without interruption on code deleted by another, or save breaking changes in a hidden branch until later. It was also immensely evil: you could hide an ancient curse or a bad word and nobody would ever know.

These two basic operations provided more complex code layout as well. Simple combinations led to nontrivial results:

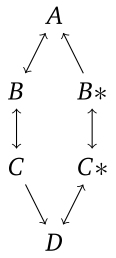


Figure 4: First, a new branch is created off of the middle two lines. Then only the latter line is merged back into the original layout. The result is a graph with no canonical representation: the text file reads completely differently depending on if it is viewed from the first line or the last.

The development of non-Euclidean text files showed exciting promise in the field of evil (think of inescapable textual labyrinths) as well as the field of interactive fiction (think of inescapable textual labyrinths). Unfortunately, the non-linearity of the files ultimately complicated PDF export, so the project was scrapped.

4 Edwin

Our final model, Edwin, uses a hybrid technique, combining the utility of edMUnD's branching mechanism and the stability of linear editing buffers. Inner details of the model and challenges faced in Edwin's development follow.

4.1 Buffers & lines

Edwin's primary organizational structure is the buffer. Each buffer is a self-consistent doubly linked list of lines. New buffers are created during branching operations, either explicitly via the branch command or implicitly by other editor operations (e.g. the deletion of a range of lines). Edwin has a trick up its sleeve that lets it parallel the smooth branching of edMUnD: every buffer contains links to the lines that were before and after it prior to branching. These links can be addressed directly or used in commands: the patch command, for example, attempts to place a buffer back into the original context of its old buffer. The explicit formalization of these boundaries saves the headache of accidentally creating paradoxical structures. Commands involving these patch links fail when the context is destoyed (i.e. if the lines around the original content are moved into different buffers and no longer have a path between them).

Lines in Edwin have unique identities; wherever possible, operations will retain them when modifying their contents or moving them around. This has the nice effect of retaining the positions of "heads" (representations of users in a channel), allowing text editing to go uninterrupted even if users remove or rearrange text in the middle of an operation.

4.2 Line numbering

The classic `ed(1)` provides one important mechanism for preventing destructive mistakes: line numbering. A cowardly user might wish to verify a range (e.g. `?^[fe]?,//`) is correct before deleting its contents; to check it, they can first use the `n` command to view the contents and line numbers of the addressed range, then use the line numbers directly to safely perform the operation. But in a collaborative environment, line numbers can silently change in between the two commands, leading to results as disastrous as getting the address wrong the first time. Edwin solves this by introducing two special line markers for every head called `<` and `>`, which always point to the first and last lines addressed in the previous command. Using `'<,'>` instead of line numbers, a user can address exactly the same lines as before.

4.3 History

`Ed(1)` implements one layer of editing history. Users can undo the last edit done and no more. Modern IDEs feature unbounded history and occasionally more complex time-traveling branch systems. Edwin compromises by implementing zero layers of history. Traditional undo methods were found to be complicated by the multitude of simultaneous edits in different locations. Rather, the function of an edit history is served flexibly by the message archive of the IRC channel itself, containing both edits and out-of-band annotations.

4.4 Extensibility

As part of a full Unix environment, `ed(1)` provides features external to the editor via a command which processes text through the shell. This is powerful in a Unix context but generally disconnected from the networking system Edwin resides in; instead, the recommended method for scripting Edwin takes place over IRC itself. Chat bots can interface with Edwin directly, inspecting data and modifying as appropriate. Given access to the editor command interface, IRC-based bots have *more* power than traditional shell scripts, as they can manipulate not only text but the state of the entire editor.

4.5 Incompleteness

The primary barrier to the use and analysis of Edwin is that I didn't finish implementing it. In order to continue research, we construct a model for what Edwin would look like and use a technique known as "guessing" to generate precise thought experimental data. We avoid bias introduced by the variance in data collection by applying similar techniques to all test environments used in the editor's evaluation.

5 Comparison with other IDEs

How does Edwin fare against other development tools? We evaluate a set of editor environments according to four criteria: overall efficiency, appearance, synergy, and evil. As a baseline for Docs and Edwin, our primary contenders, we evaluate two classic editor environments: Vim over a paper teletype and Microsoft Paint over VNC.

5.1 Efficiency

While basic writing operations are simple in Paint, moving and replacing text requires manual intervention and is prone to failure due to Paint's relatively basic addressing system. All of these complications are amplified by the time required for a full screen refresh. Paint's ineptitude is only second to that of Vim, in which every keystroke sends dozens of mangled control characters to the poor, poor teletype output.

Docs does its job great: writing text has a noticeable network delay, but visual changes render fast on individual client machines. It supports a wide range of useful operations and has no problem restructuring large files. Edwin performs comparably if slightly better: feedback is only sent when explicitly requested, and moving from a granular editing system to a line-based one permits the correction of quick mistakes without network delays. Edwin's range of operations is similar to that of Docs, but is better inclined to complex organization with its native marks, buffers, and regexen.

5.2 Appearance

The appearance of code written in Paint is entirely dictated by its author. Paint has 24-bit color capability; its main limiting factor is the ability of the user to not just handwrite but handwrite *with a mouse over VNC*. Vim is somehow worse.

Docs has native rich text support, which permits users to highlight their code's syntax however appropriate. However, its font selection is limited: only the Google Fonts repository is available. Edwin supports rich text by means of IRC's formatting control characters and an unlimited range of fonts provided by chat clients.

5.3 Synergy

There is nothing more synergetic than a shared whiteboard. Paint is only limited by its single shared drawing cursor. Vim has no synergy at all—even if it were comprehensible, everyone'd have to crowd over the teletype. Might as well just draw on the paper.

Docs is at the forefront of channeling synergy through text: it supports the core components of group collaboration, with live editing, comment threads, and edit proposals. But it fails to recover synergy when it is lost—for example, when two simultaneous edits come into conflict, competing with one another for space in the finished document. Edwin supports all of Docs' features through a unified chat interface, and its branching mechanism allows for peaceful resolution of conflicting simultaneous work.

5.4 Evil

Paint supports evil: you can draw crude pictures wherever you want and, since its undo stack is limited, force your collaborators to either suffer through the drawings or completely erase whatever they intersected. Vim doesn't seem to support evil: certain Vim reimplementations in other environments are rumored to include it but were not considered for evaluation.

Google was initially reluctant to support being evil but faced pressure to allow large enterprises to use the quickly growing Docs; it eventually removed its evil restrictions in 2018. While now seeing broad use in evil, the editing interface is anything but: any malicious changes can be found and undone with its thorough history tracker, and drawing unsightly pictures in regular text documents is tedious. Edwin has supported evil since its conception, in both its use and its internal operation. It is easy to not only write swear words but irreparably overwrite entire documents, exponentially increase the editor's used memory, and cause the bot to flood any IRC channel naïve enough to associate with it.

6 Future work

I sure hope it d

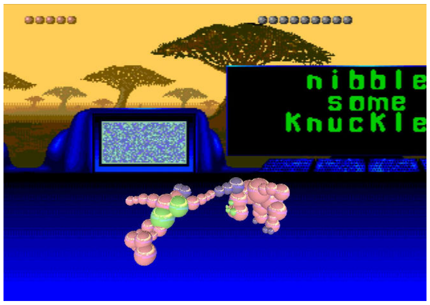
7 Future work

It would be nice if the editor existed. Additionally, Edwin's realization lacks a system for traditional, long-term collaboration. One potential model would use email: users could send files containing Edwin commands to a project maintainer, who would apply the commands to a central repository.

References

[1] IEEE and The Open Group. 2018. ed – edit text. The Open Base Specifications Issue 7, 2018 edition. https://pubs.opengroup.org/onlinepubs/9699919799/utilities/ed.html

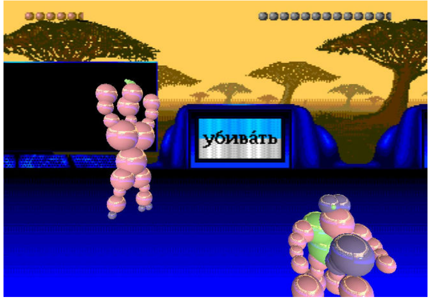
w
quit
q
^C



Did you think Ballz 3D for SEGA Genesis was an amazing experience? Ballz 3D 3D is **twice as 3D,** thanks to NVidia RTesque graphics.

BAL
OUT

Get ready for a game that plays more-or-less exactly the same, now **with reflections.**



Screenshots shown may not match final cartridge.

The soundtrack is bad in a 90s way. Not "bad" like the power glove was purported to be, but bad like the power glove *was*.

LLZ

-ER.

Exclusively on
SEGA 32esque
on SEGA 32X
on SEGA CD
on SEGA Genesis.

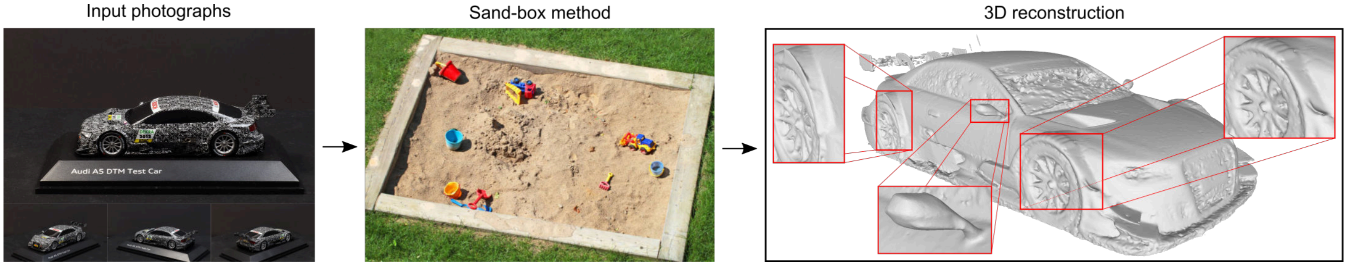
Produced by walkingsep, tokkot, and ix. Contact ix@tchow.com for details.

A Free Computer Vision Lesson for Car Manufacturers
or
It is Time to Retire the Erlkönig

Maximilian Weiherer   Bernhard Egger

Fantastic-Amazing-University Erlangen-Nürnberg (FAU)

maximilian.weiherer@fau.de
bernhard.egger@fau.de



Oh no – we can recover very detailed 3D shape from Erlkönig photographs. How could this have happened? And even worse, the reconstruction is best in the parts where there is a pattern. Well, we guess someone didn't know about the basics of computer vision.

Abstract

In award winning prior work [1], we identified the inability of autonomous cars to honk as the key reason that they are not broadly deployed on our streets. In this work [2], however, we suggest that the core reason is the lack of most basic computer vision knowledge of car manufacturers. To hide their most fancy new cars they put a special camouflage pattern on their so called *Erlkönig* prototypes. The pattern is designed to trick our perception; at the same time it enables computer vision systems to perfectly recover the 3D shape of the prototype – even better than without the pattern as we show in this paper. How could we expect a prototype car that already demonstrates a lack of computer vision knowledge to ever evolve into an autonomous vehicle?

1. Introduction

We could now tell you the whole story of the Erlkönig and Dazzle Camouflage patterns, but that is way to much work. The interested reader (if any) is kindly asked to read the relevant literature here: https://en.wikipedia.org/wiki/Dazzle_camouflage. The relevant portion is that the pattern is designed to make it hard to estimate the range, speed and heading of a ship and it might also make it harder to estimate the type of the ship, see Figure 1. And this is all cool and everything, but over 100 years have passed since the pattern was described by Norman Wilkinson during the last pandemic. This does not hold back car manufacturers to paint their dinosaur eating machines (also the sun eating counterparts) with patterns motivated by this idea and brag about it (https://www.bmw.com/de/automotive-life/erlkoenig-auto.html). The car manufacturers might try to hide the shape of their cars, and it might work pretty well when it comes to the human eye. It might even hold some cameras back from using their automatic focus. But all of that only holds for a single view! The moment we have multiple viewpoints – tadaaa – we can use the whole ballpark of computer vision algorithms and even algorithms from the stone-age (all historic works before 2022) of computer vision lead to an almost perfect 3D reconstruction.

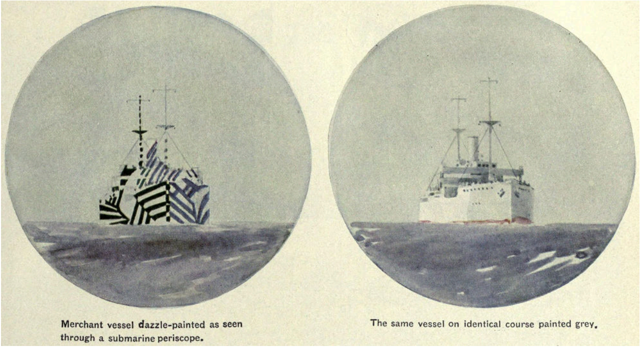
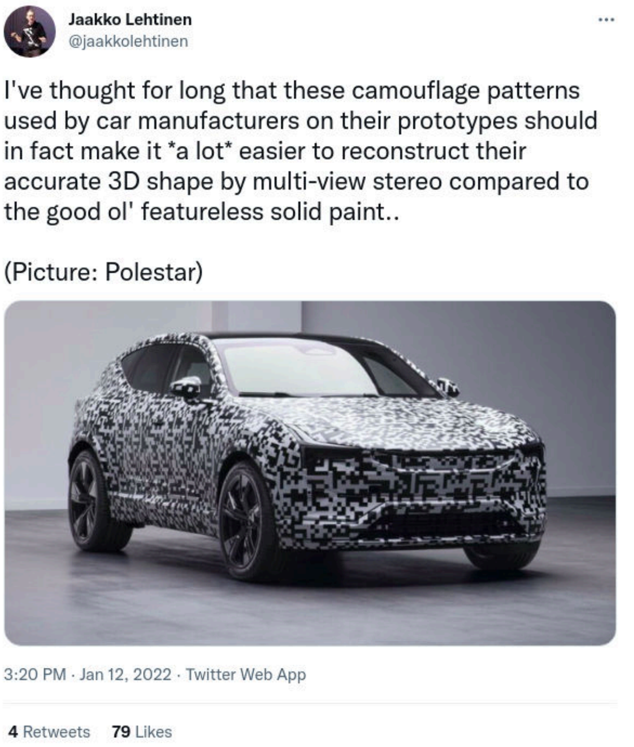

Merchant vessel dazzle-painted as seen through a submarine periscope.
The same vessel on identical course painted grey.

Figure 1. Damn hard to estimate the heading of the boat on the left with the Dazzle camouflage pattern, right? Source: Encyclopædia Britannica, 1922 / Wikipedia.

To summarize, in this paper, we impressively show that the 3D shape of a car covered with camouflage patterns (i.e., an Erlkönig) can be very well reconstructed just from a set of ordinary photographs, taken from different perspectives. To make the embarrassment for car manufacturers perfect, we demonstrate that the 3D reconstruction of the same car *without* patterns is much worse. That's bitter.

1.1 Related Work

Besides a Twitter thread with various researchers almost scooping us there is no relevant prior work (see Figure 2).



Jaakko Lehtinen
@jaakkolehtinen

I've thought for long that these camouflage patterns used by car manufacturers on their prototypes should in fact make it *a lot* easier to reconstruct their accurate 3D shape by multi-view stereo compared to the good ol' featureless solid paint..

(Picture: Polestar)

3:20 PM · Jan 12, 2022 · Twitter Web App

4 Retweets 79 Likes

Figure 2. Almost scooped: https://twitter.com/jaakkolehtinen/status/1481269802681393153.

2. Methods

Given a set of photographs taken from different perspectives, we used BASIC photogrammetry for 3D reconstruction, see e.g. https://en.wikipedia.org/wiki/BASIC. The choice fell on BASIC because we are absolute beginners when it comes to programming (and all this computer stuff in general). Following common practice and due to the lack of knowledge, we do not reveal all the details about our method. But we want to sprinkle in some unnecessary details like for example the parameters

$$\xi \quad \text{and} \quad \zeta \tag{1}$$

that are not introduced but very critical and car-fully set to the value 5. As far as we understand, a key step in our pipeline involves the solution of the *Perspective-n-Point* (P-NP) problem. Having a solution of the P-NP problem at hand and exploiting the fact that $\mathrm{P} = \mathrm{NP}$[1], obtaining the final 3D reconstruction in form of a triangular surface mesh is dead easy and no more information is needed to re-implement this.

3. Experiments and Results

We took 32 photos of an Erlkönig[2] and applied the method described earlier to obtain a 3D reconstruction. About the same procedure was used to reconstruct the same car *without* a pattern. Everything went perfect on the very first try (yes, no testing or training or what else was needed) and our method produced spectacular results, see Figure 3. As expected by computer vision experts, the reconstruction of the Erlkönig is at least a million (10^6) times better than the reconstruction obtained from the same car without a pattern. These results are so good that we don't even need a quantitative evaluation, right?

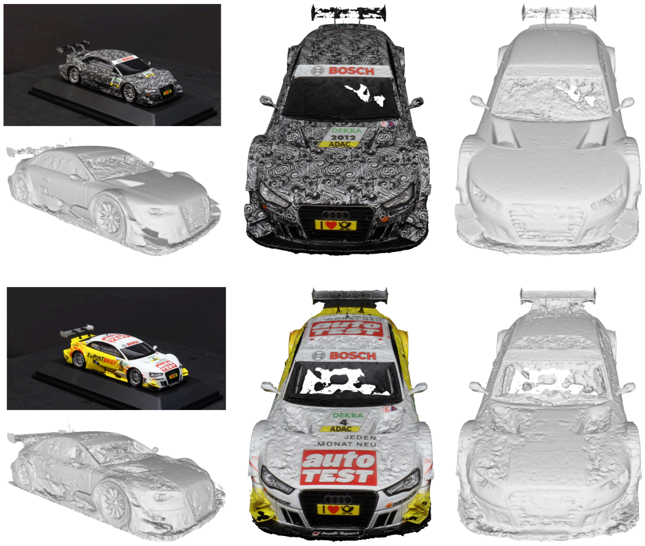


Figure 3. See, pattern leads to awesome reconstruction whilst no pattern no good. Please print in grayscale on dead trees for better visibility of details.

4. Limitations

Car manufacturers will say we are missing something here and things are more complex. Don't believe them. They also say burning dinosaurs is cool and that we will have autonomous cars next year! We win, they loose – it's that easy.

5. Conclusion

To summarize, the pattern might be able to cause a collision with a Tesla, but for any other purpose it is beyond repair: the Erlkönig is asking for retirement – loud and clear. Just like this Tim Brady!

Our future mission should be clear and obvious: reveal the shape of all the things in this world covered with a camouflage pattern (or something that looks like a camouflage pattern; doesn't matter, our method will tackle it anyway because it generalizes). This immediately leads to the following research questions: what the heck is really hiding under a military uniform? And, can we trust QR codes? Stay tuned and make sure to follow us on Twitter.

Finally, if you want to learn more about computer vision: there might be a cool lecture at your favorite astonishing university (FAU). If you are a car manufacturer: we have some ideas to help you out of your misery and are looking for funding.

References

[1] B. Egger and M. Siegel. Honkfast, prehonk, honkback, prehonkback, hers, adhonk and ahc: the missing keys for autonomous driving. *SIGBOVIK*, 2020.

[2] M. Weiherer and B. Egger. A free computer vision lesson for car manufacturers or it is time to retire the erlkönig. *SIGBOVIK (under careful review by very talented, outstanding reviewers)*, 2022.

[1] A proof is provided in the Appendix. In essence, however, the ingenuity in our proof was to use the well-known fact that $e^{i\pi} - 1 = 0$ <MW: Damn, just realized we've been using this formula wrong all the time. BE: Relax. No one will notice.>. Kudos to Leo Euler. @ClayInstitute: We expect the money no later than May 1st, 2022. We need it. Desperately.

[2] Due to an ongoing legal dispute, we unfortunately cannot share any details about the car. Please refrain from e-mail inquiries. Authors may not have access to the internet for an unknown period of time.

Redundant Coupling

Peter Kos
Rochester Institute of Technology
plk5062@rit.edu

Abstract

In the field of software engineering, researchers have coined the terms “cohesion” and “coupling” to describe the structure and interaction between classes in an object-oriented environment [6]. The traditional wisdom is to use low coupling and high cohesion to reduce side-effects while making isolated changes in code. In this paper, we present an alternative methodology: *redundant coupling*. In redundant coupling, coupling between classes is maximized in order to create the strongest possible foundation in the architecture. This has been shown to provide a 45% increase in unit test stability (in flaky environments), 71% increase in developer confidence, and a 14% increase in my personal happiness since my second wife left me.

1 Introduction

Coupling and cohesion are core ideas in software architecture. They are a little ambiguous too, as modern computer science education does not usually address the issues that can (allegedly) arise from high coupling and low cohesion.

2 Theory

Coupling is introduced as a way to “[minimize] the paths along which changes and errors can propagate into other parts of the system, thus eliminating disastrous ‘ripple’ effects” [6, p. 233]. However, this operates on two false assumptions:

1. Individual change’s effects on the whole system need to be minimized
2. Errors can propagate into other parts of the system

Changing a piece of code involves a deep insight into its function. (One can imagine careless changes, but these are part of the natural software development lifecycle).

...

Errors, too, are by definition, a symptom of the system. The solution for an error may be in an individual class, or across multiple classes, yet the codebase as a whole suffers.

2.1 Definitions

Definition 1. *Equivalent access is a reciprocal access between two classes A and B, such that the following holds:*

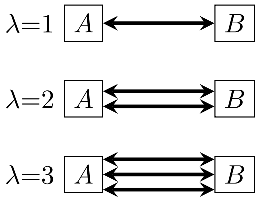
- *Let α be a field of A,*
- *Let β be a field of B,*
- *Let $\mathbb{A}(\alpha)$ be an access of field α,*
- *Let $\mathbb{A}(\beta)$ be an access of field β*

An access is equivalent if and only if

$$\Theta(\mathbb{A}(\alpha)) = \Theta(\mathbb{A}(\beta))$$

- For the fields $\alpha =$ `String` and $\beta =$ `String`, these would be an equivalent access, as the access/set complexity of a string [1] is $\mathcal{O}(1)$.
- For the fields $\alpha =$ `HashMap<T>` and $\beta =$ `HashMap<T>`, the same is true, as here, the complexity of any hash map operation is $\mathcal{O}(n)$.
- For the fields $\alpha =$ `String` and $\beta =$ `HashMap<T>`, $\mathbb{A}(\alpha) = \mathcal{O}(1)$, and $\mathbb{A}(\beta) = \mathcal{O}(n)$. (A HashMap, with a sufficiently large load factor, may need to iterate through all of its elements to get a specified ith element.) Therefore, in this case, class A and B would **not** have equivalent access between fields α and β.

Definition 2. *The coupling factor λ is defined to be the number of equivalent accesses between two classes A and B.*



[1] In most programming languages.

2.2 λ coupling factor

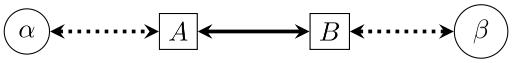
We define two classes A and B to have *one-string coupling* if they are bidirectionally coupled in one mechanism, whether that be:

- Reciprocal method-calling, or
- Equivalent access.

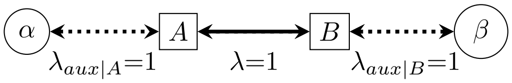
To see how redundant coupling increases the foundational stability of a class, we take to the example of auxiliary classes.

Auxiliary classes are defined as classes that we do not anlayze at the present moment, that are **not** coupled to any other class. The load factor between two classes A and B, e.g., λ_{AB}, is only defined with respect to A and B. (Later on, we define a process called promotion that addresses how we can move between classes of focus).

Say that we have an auxiliary class α that is coupled to A, and the same with β for B.



We can see that the load factor between each class is as follows:



The auxiliary coupling factor, $_{aux}$, is equal to the coupling factor of our two classes of interest (λ). We define this to be a point of **Fowler Equlibrium**.

Definition 3. *A* ***Fowler Equilibrium*** *is present between two classes A and B such that*

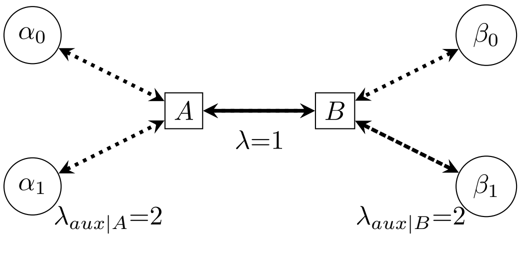
$$\lambda_{AB} = \lambda_{aux|A} = \lambda_{aux|B}$$

(where λ_{AB} is the coupling factor between A and B, and $\lambda_{aux|A}$ is the coupling factor between the auxiliary classes of A.)

The total auxiliary coupling factor is represented as such:

$$\sum_{\alpha \in A} \lambda = \lambda_{aux|A}$$

However, once we add two additional auxiliary classes on each side, we run into an issue:



Here, the auxiliary coupling factors are not at equilibrium with the interior coupling between our classes of interest. This ia a **misbalanced** system, and this is fundamental issue this paper serves to address.

3 Limitations

We acknowledge [2] that it is "unlikely, for financial and structural reasons" [5] to develop a model that can account for coupling in non-OOP paradigms.

Additionally, it is not known how this would affect projects in multiple languages. Mayer and Schroeder define an `XLL approach` [3, p. 97], where developers manually specify links between abstract concepts present in each language. In this system, one could couple the artifacts themselves, but this is more akin to coupling specific fields within a class – not the class overall.

4 Testing Methodology

Tests were originally written using a source-compiled version of JUnit 2, until the inferiority of this solution was realized through several cognitive behavioral therapy sessions.

Instead, we pivoted to use `Rust 1.58.0-nightly`. Rust [2] provides a safe, fast environment without garbage collection, which allowed us to mitigate any cross-bag insect contamination [4].

One potential issue is a common misconception that Rust does not provide an OO environment. We assert that it is possible to represent an OO-pseudostructure within Rust, without resorting to an inferior language. (See Appendix A.)

5 Results

As shown previously [1], we found a few notable improvements in codebases that used redundant coupling over the traditional low coupling / high cohesion model:

1. 45% increase in unit test stability
2. 71% increase in developer confidence
3. 38% reduction in alimony paperwork

Unit test stability occurred a natural consequence of allowing changes to propagate throughout the codebase. Intermittent tests can be caused by any number of bugs, and are usually localized to one or two classes. Through redundant coupling, the codebase is unilaterally influenced by these issues, which allow the system overall to behave more reliably.

Developer confidence was also observed to increase markedly.

[2] i.e., forced to concede

6 Conclusion

Todo.

7 Appendix A

```
// Some sample code we wrote as part of a
    demonstrative paper on Rust coupling.
// We are not sure if it compiles.

struct User {
    var first_name: &str,
    var last_name: &str,
    var age: u8
}

impl User {
    fn get_full_name(&self) -> &str {
        first_name + last_name
    }
    fn have_birthday(&mut self) {
        self.age += 1;
    }
}

struct Login {
    var cur_user: User
}

impl Login {
    fn login(&mut self) {
        todo!();
    }
    fn get_full_name() -> &str {
        cur_user.get_full_name()
    }
}
```

Listing 1: Example of an OO structure in Rust, with coupling

References

[1] Peter Kos. Redundant coupling. Association for Computational Heresy, 2022.

[2] Nicholas D. Matsakis and Felix S. Klock. The rust language. In *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*, HILT '14, page 103104, New York, NY, USA, 2014. Association for Computing Machinery.

[3] Philip Mayer and Andreas Schroeder. Cross-language code analysis and refactoring. In *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*, pages 94–103, 2012.

[4] Sulochana Paudyal, George P Opit, Frank H Arthur, Georgina V Bingham, Mark E Payton, Sandipa G Gautam, and Bruce Noden. Effectiveness of the zerofly® storage bag fabric against stored-product insects. *Journal of stored products research*, 73:87–97, 2017.

[5] Michael L. Scott. Cover letter for dean of the golisano college of computing and information sciences. 2022.

[6] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Systems Journal*, 38(2):231–256, 1999. Copyright - Copyright International Business Machines Corporation 1999; Last updated - 2021-09-10; CODEN - IBMSA7.

Theory

13 Destructive Logic

Runming Li

Keywords: logic, constructive logic, destructive logic, type theory

14 Overlap-Maximal Graph Labelings: Graph Labelings with Non-Disjoint Vertex and Edge Sets (and how they can be used for encryption, poetry, and breaking mathematics)

Gabriel Chuang

Keywords: Graph Theory, Cryptography, Fox in Socks by Dr. Seuss

15 Neo-classical Logic: Une Logique non classique

Martin Vassor and Fangyi Zhou

Keywords: Logic, Meta-Metatheory, Law of Excluded Middle, Law of Contradiction, Decidability, Macron

16 A Patriotic Analysis of Programming Paradigms

Jacob Weiner

Keywords: Patriotism, Constitution, Dependent Type Theory

17 On Ruinment: Ruination Theory and its Consequents

Luna Tascheter

Ruination Theory, Ruinment, T0T4L PWN4G∃, Category Theory, Adam Conover, Adam Sandler, Hallmark, Waste of Paper, Orange

Destructive Logic

Runming Li
runmingl@andrew.cmu.edu
Carnegie Mellon University

March 19, 2022

Abstract

Intuitionistic logic, also know as constructive logic, is considered the true logic of computer science for the reason that it contains actual computational content. We propose Destructive logic, a supplement of constructive logic, with the goal of proving things that are not provable in constructive logic. Destructive logic, suggested by its very name, proves things by destroying, rather than construction. We study its properties and applications in this paper.

0 Introduction[0]

Constructive logic was historically proposed in response to classical logic. Everything that is provable in constructive logic is provable in classical logic, but not vice versa (e.g. law of excluded middle). In this sense, constructive logic is a more restricted, more precise logic than classical logic. It was later found out that constructive logic has roots in type theory and programming language theory based on Curry-Howard Isomorphism[1]. Constructive logic is so well-studied that many major universities teach it. Among them one of the classical[2] courses is 15-317 Constructive Logic at Carnegie Mellon University.

Duality exists everywhere in logic: conjunction and disjunction, universal and existential, left and right, up and down, you and me. A natural question to ask is: what is the dual of constructive logic? An unsurprising answer from etymology suggests destructive logic. Indeed this is the case: while constructive logic proves things by providing a clear algorithm that constructs the object in question, destructive logic proves things by destroying things it wants to prove. Of course, unreasonable destruction is considered psychopath, so we need to develop a clear justification for every destruction. For the purpose of destructive logic, introduce 🦖, named *Destro* (they/them/theirs), a rational T-rex whose main purpose of life is to destroy things they do not like.

[0] It is well-known that a good logician starts counting from 0.

[1] Also know as *Propositions as types* principle. Also know as *Programs as proofs* principle. Also know as *Brouwer-Heyting-Kolmogorov interpretation*. Also know as *Realizability interpretation*. Also know as *Curry-Howard Correspondence*. It is worth noting that Curry-Howard Isomorphism is in no sense an isomorphism. Just like hotdog is in no sense a dog.

[2] Classical course, in a constructive sense.

1 Extending Natural Deduction

Gentzen[2] proposed natural deduction system to formulate intuitionistic logic. We extend his natural deduction system in support of our destructive logic.

1.1 Conjunction

In constructive logic, conjunction can be summarized by the following rules.

$$\frac{\Gamma \vdash A\ \mathit{true} \quad \Gamma \vdash B\ \mathit{true}}{\Gamma \vdash A \wedge B\ \mathit{true}} \wedge I \qquad \frac{\Gamma \vdash A \wedge B\ \mathit{true}}{\Gamma \vdash A\ \mathit{true}} \wedge E_1 \qquad \frac{\Gamma \vdash A \wedge B\ \mathit{true}}{\Gamma \vdash B\ \mathit{true}} \wedge E_2$$

Destro feels uneasy about this kind of setup: everything is too deterministic, and they want to attack propositions they do not like. Admittedly Destro's personal taste is too hard to predict, so we have the following possibilities.

$$\frac{\Gamma \vdash A\ \mathit{true} \quad \Gamma \vdash B\ \mathit{true}}{\Gamma \vdash 🦖 \wedge B\ \mathit{true}} \wedge I_1 \qquad \frac{\Gamma \vdash A\ \mathit{true} \quad \Gamma \vdash B\ \mathit{true}}{\Gamma \vdash A \wedge 🦖\ \mathit{true}} \wedge I_2$$

$$\frac{\Gamma \vdash 🦖\ \mathit{true} \quad \Gamma \vdash B\ \mathit{true}}{\Gamma \vdash A \wedge B\ \mathit{true}} \wedge I_3 \qquad \frac{\Gamma \vdash A\ \mathit{true} \quad \Gamma \vdash 🦖\ \mathit{true}}{\Gamma \vdash A \wedge B\ \mathit{true}} \wedge I_4$$

$$\frac{\Gamma \vdash A\ \mathit{true} \quad \Gamma \vdash B\ \mathit{true}}{\Gamma \vdash 🦖 \wedge 🦖\ \mathit{true}} \wedge I_5 \qquad \frac{\Gamma \vdash 🦖\ \mathit{true} \quad \Gamma \vdash B\ \mathit{true}}{\Gamma \vdash 🦖 \wedge B\ \mathit{true}} \wedge I_6$$

Again, Destro's destruction is non-deterministic (at least humans should never be able to learn a T-rex's personal opinion), so we would not bother writing out all the possibilities in which they can attack for the purpose of saving some space. Moreover, one day Destro is tired of destructing propositions, so they start to destroy rules and context and even turnstile.

$$\frac{🦖 \vdash 🦖\ \mathit{true} \quad \Gamma 🦖 🦖\ \mathit{true}}{\Gamma \vdash 🦖 \wedge 🦖\ \mathit{true}} \wedge 🦖$$

1.2 Disjunction, Implication, Truth, and Falsity

Now one can simply img🦖ine what hap🦖en when we decide to put 🦖into other con🦖ectives such as disjunc🦖ion, implic🦖tion, truth, and falsi🦖y. In short, they just destroy whatever they want.

2 Properties and Applications

2.1 Basic Theorems

We first prove some basic theorems of destructive logic to convince the readers that indeed destructive logic is a real logic.

Theorem 1. *Consistency: it is not the case that $\perp$ true.*

Proof. In Curry-Howard Isomorphism, $\perp$ corresponds to $\mathbf{0}$ the empty type. 🦖cannot resist destroying it because it looks so much similar to their baby egg. Therefore, whenever the prover is able to show $\perp$ *true*, 🦖destroys it. □

Theorem 2. *Local completeness of conjunction: the elimination rules of conjunction are not too weak.*

Proof. Imagine a day when 🦖is too tired and they decide not to destroy anything. Then for that day destructive logic downgrades to constructive logic, in which local completeness of conjunction holds. □

Corollary 2.1. *Local completeness of other connectives.*

Proof. Copy and paste the proof of Theorem 2. □

Theorem 3. *Local soundne🦖s of conjunction: the elimination rules of conjunction are not too strong.*

Proof. Local soundne🦖s has been destroyed by 🦖. □

Theorem 4. *Global completeness and global soundness in destructive logic.*

Proof. The author has been destroyed by 🦖, and therefore leave the proof of this theorem to readers. □

2.2 Curry-Howard Isomorphism

Behind every logic there is a type system, according to Curry-Howard Isomorphism: constructive logic has simply typed lambda calculus, classical logic has continuation, linear logic has linear type system, second order logic has system F. It is only natural that destructive logic also has a corresponding type system. For this purpose, we propose υ-calculus[3]. We extend simply typed λ-calculus with the following types and terms.

$$\tau ::= \cdots | \text{🦖} | \tau^{\tau} | \texttt{unknown\_destroy}$$

$$e ::= \cdots | \text{🦖} | \upsilon(e, e) | \texttt{destro}(e)$$

The statics would be as followed.

$$\frac{}{\Gamma \vdash \text{🦖} : \text{🦖}} \qquad \frac{}{\Gamma \text{🦖} e : \texttt{unknown\_destroy}} \qquad \frac{}{\text{🦖} \vdash e : \texttt{unknown\_destroy}}$$

$$\frac{}{\text{🦖🦖} e : \texttt{unknown\_destroy}} \qquad \frac{}{\text{🦖🦖🦖🦖🦖}} \qquad \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \upsilon(e_1, \texttt{destro}(e_2)) : \tau^{\text{🦖}}}$$

[3] The choice of greek letter υ here is not arbitrary, but rather forced. Type theorists have used up so many good greek letters: α is for equivalence, β is for reduction, γ is for substitutions, Π and Σ are for dependent types, δ and o are for Covid, to name a few.

In this calculus, when 🦖accidentally destroy something strange (such as context or turnstile), we simply give it a universal type of `unknown_destroy`. The essence is the υ expression, where if e_1 is the undestroyed expression and e_2 is its destroyed equivalence, then $\upsilon(e_1, \texttt{destro}(e_2))$ is the way to recover the computational meaning of what 🦖destroyed. Then it follows naturally that

$$\cdot \vdash \upsilon(\upsilon(\text{🦖}, \texttt{destro}(\text{🦖})), \texttt{detsro}(\upsilon(\text{🦖}, \texttt{destro}(\text{🦖})))) : \text{🦖}^{\text{🦖}^{\text{🦖}}}$$

2.3 Application

We suggest that destructive logic and υ-calculus be used to analyze randomized algorithms, since 🦖's destruction pattern is essentially randomized.

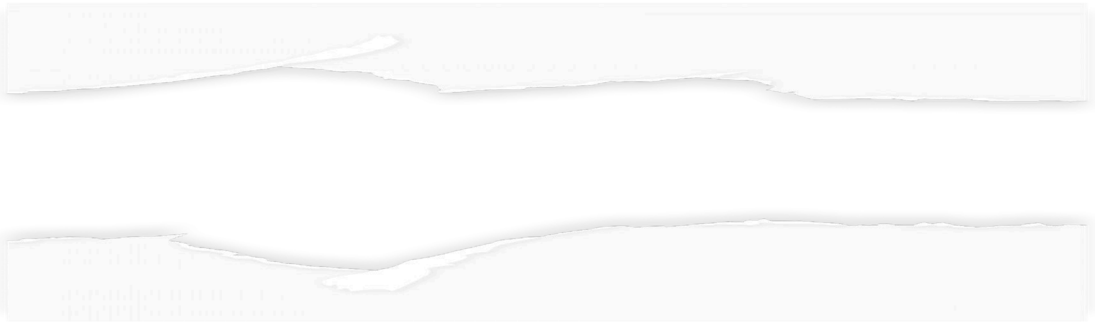
3 Future Work

We urge the logic community to take destructive logic seriously and put drastic amount of work in studying it. We further propose that destructive logic be taught at major universities. For Carnegie Mellon University, we propose to start a new course 15-407 Destructive Logic, and place it in the Principles of Programming Languages Concentration requirement. The choice of course number here is not arbitrary: according to Chuang and Wu[1], in order for a course to be considered a PL course, it has to have a Collatz number of 59. Indeed, 15407 has a Collatz number of 59 and should be considered as a PL course. Moreover, every course with a 7 in the course number is a good course (e.g. 15-317, 15-417, 98-317). It follows naturally that the course mascot should be 🦖, named *Destro*.

References

[1] Brandon Wu Gabriel Chuang. "What Lothar Collatz Thinks of the CMU Computer Science Curriculum". In: *SIGBOVIK 2021* (2021).

[2] Gerhard Gentzen. "Untersuchungen über das logische Schließen. II". In: *Mathematische Zeitschrift* 39.1 (1935), pp. 405–431. DOI: `10.1007/bf01201363`.



This page has been destroyed by 🦖.

Making Graphs Really Hard to Say Out Loud

Graph Labelings with Non-Disjoint Vertex and Edge Sets

and how they can be used for encryption, poetry, and breaking mathematics

Gabriel Chuang (gtchuang@andrew.cmu.edu)
Carnegie Mellon University

***Abstract*— Modern communication systems have greatly increased the ease with which we can describe our ideas to others. In an effort to slow this progress and hinder scientific advancement, we examine labelings of graphs that *reduce* clarity by reusing edge labels for vertices. We bound how much overlap can be achieved, show how such labelings are cryptographically secure, and prove some theorems that suggest that all of mathematics is broken. Finally, we present some applications to poetry and propose a new naming convention for graphs based on these labelings, which we hope will be widely adopted.**

I. Introduction

In Chapter 1.1 of Diestel's widely-used textbook *Graph Theory*, a graph is defined to be a pair of sets (E, V) with $E \subseteq [V]^2$. Diestel follows this up with the following note:

> To avoid notational ambiguities, we shall always assume tacitly that $V \cap E = \varnothing$.

Needless to say, this is very disappointing to fans of notational ambiguity. There are many settings where being notationally ambiguous is beneficial, such as:

- When brevity is of the essence (e.g. when lecturing at a blackboard),
- When trying to sneak faulty proofs past reviewers,
- When trying to confuse students, and
- When trying to confuse yourself.

In this work, we will explore labelings of graphs in which we seek to maximize the overlap between the labels of the vertex set V and edge set E. (Note that we will continue to require that edges are two-element subsets of the vertices). See Fig. 2 for an example of such a graph. Note that that graph has an edge $\{a, b\}$, but also a *vertex* $\{a, b\}$!

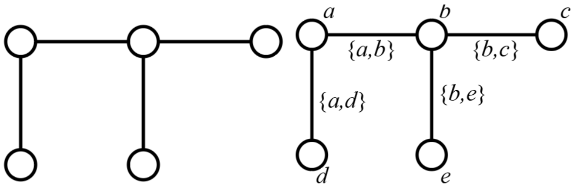
Such labelings have many desirable attributes; most notably, graphs represented in this way are extremely notationally ambiguous, which can allow us to prove interesting theorems. They are also very hard to say out loud, due to repetition in their vertex and edge sets.

These labelings may also be of interest to those seeking to keep their graphs concealed from adversaries who can only communicate verbally. We present a cryptographic protocol that is in fact secure against *all* adversaries.

Finally (and of purely academic interest)[1], such labelings of graphs often resemble rhyme schemes; we present several graphs whose edge sets correspond to common rhyme schemes or famous poems, and propose an alternative naming system based on poetic forms.

II. Motivation

Consider the following graph, and a natural labeling of its vertices:



(a) A simple undirected graph G.

(b) A "natural" labeling of G, with vertex set $V = \{a, b, c, d, e\}$.

Fig. 1: A simple undirected graph, and a natural labeling.

Unfortunately, this labeling is extremely boring: there is no overlap between V and E! Instead, consider the following labeling:

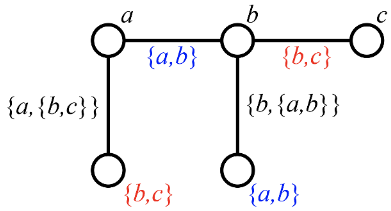


Fig. 2: A much more interesting labeling of the graph. The vertex set is $V = \{a, b, c, \{a, b\}, \{b, c\}\}$.

[1] just like the rest of this paper, to be clear

We've managed to achieve an overlap of size 2: $V \cap E = \{\{a, b\}, \{b, c\}\}$. That is, there are two vertices that share a label with an edge. Can we do better on this graph? As it turns out, we can:

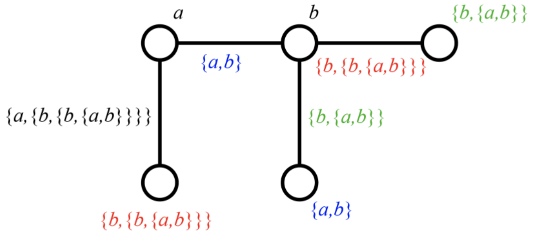


Fig. 3: An overlap-maximal labeling of the graph. The vertex set is $V = \{a, b, \{a, b\}, \{b, \{a, b\}\}, \{b, \{b, \{a, b\}\}\}\}$. In this case, three of four edges share a label with a vertex.

This labeling of the graph is has many desirable properties. For example, the verbalization is extremely ambiguous:

> "Vertices: A, B, A, B, B, A, B, and B, B, A, B. Edges: A to B, B to A, B, B to B, A, B, and A to B, B, A, B." [2]

In general, it is not obvious how much overlap can be achieved for arbitrary graphs. Fig. 3 achieves overlap on 3 of 4 edges; can one do better? What classes of graphs enable maximum overlap?

A central point of this paper will be to carefully define such labelings and to show upper and lower bounds on how much overlap can be achieved.

III. Definitions and Central Claim

Definition 1 (Overlap number)**.** *For a graph $G = (V, E)$, let the overlap number $\psi(G)$ be the maximum size of $E \cap V$ over all possible labelings L of G, i.e., the maximum number of edges that can share a name with a vertex (or vice versa). That is,*

$$\psi(G) = \max_{L:V \to \mathbb{U}} |L(E) \cap L(V)|.$$

Definition 2 (Overlap ratio)**.** *The overlap ratio $\rho(G)$ of a graph $G = (V, E)$ is defined to be*

$$\rho(G) = \frac{\psi(G)}{|E|}$$

In the example graph of Fig. 3, $\psi(G) = 3$ and $\rho(G) = \frac{3}{4}$.[3]

Theorem 1. *For any graph $G = (V, E)$,*

$$\psi(G) = \min(|E|, |V| - 2).$$

In particular, since connected graphs have $|E| \geq |V| - 1$,

Corollary 1. *For a connected graph $G = (V, E)$,*

$$\psi(G) = |V| - 2$$

The proof of Theorem 1 shows that $\min(|E|, |V| - 2)$ is both an upper and lower bound for $\psi(G)$, using a partial order argument for the upper bound and an explicit construction for the lower bound. To spare you the gory details, the proof has been relegated to Appendix A.

IV. Overlap Ratio and Pretty Pictures

In this section, we will show some examples of maximum-overlap labelings of a few common classes of graphs, and discuss the overlap ratio induced by those labelings.[4]

A. Complete graphs K_n

K_n has $\binom{n}{2}$ edges. Theorem 1 implies that

$$\rho(K_n) = \frac{\psi(K_n)}{\binom{n}{2}} = \frac{2(n-2)}{n(n-1)} = \Theta\left(\frac{1}{n}\right).$$

Maximum-overlap labelings of K_5 and K_7 are shown in Fig. 4.

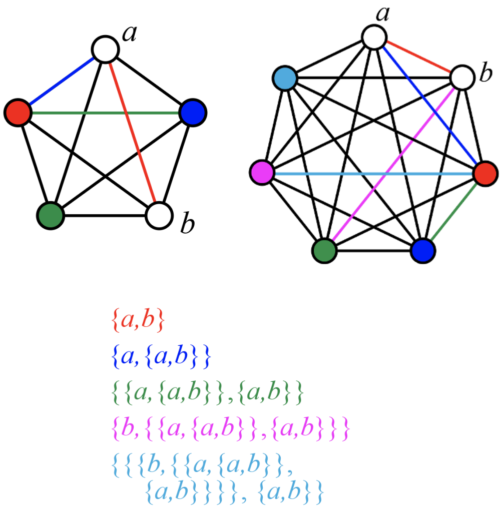


Fig. 4: Overlap-maximal labelings for K_5 and K_7. Note that very few edges are reused for vertex labels.

The edge set for a max-overlap labeling of K_7 can be found in Appendix B.

[2] We could've done even better by naming the vertices "and" and "to" instead of A and B.

[3] In the spirit of being notationally ambiguous, we will abuse notation and use $\rho(G)$ for non-maximal labelings as well.

[4] More importantly, we will have many colorful figures, to maximize appeal to children, advertisers, and color theory gremlins.

B. *Cycles C_n*

C_n has n edges. Theorem 1 implies that

$$\rho(K_n) = \frac{\psi(C_n)}{n} = \frac{n-2}{n} = 1 - \frac{2}{n} \sim 1.$$

An overlap-maximal labeling of C_n is shown in Fig. 5

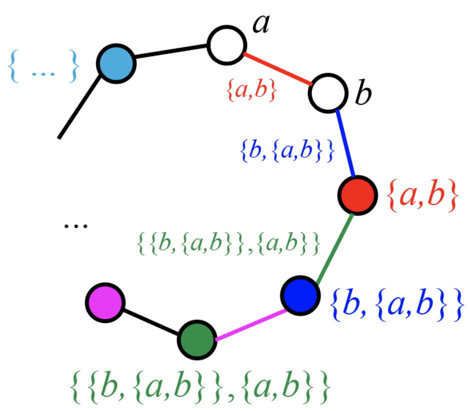


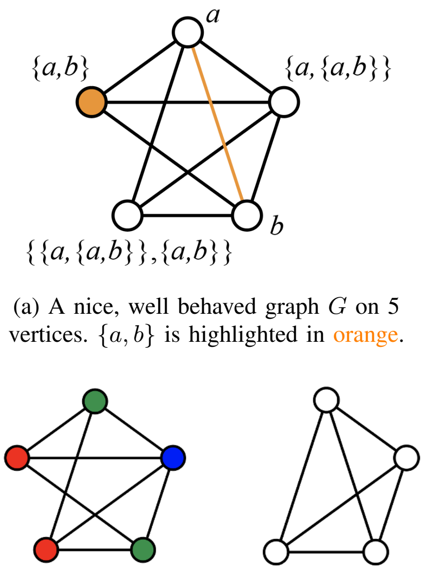
Fig. 5: Overlap-maximal labeling for C_n. Nearly every edge label is reused.

V. Notational Ambiguity Enables Chaos

Now that we have our framework for labeling graphs, we will, naturally, attempt to prove some theorems.

A. *Graph coloring*

Let G be the graph shown below in Fig. 6a.



(a) A nice, well behaved graph G on 5 vertices. $\{a,b\}$ is highlighted in orange.

(b) $G \setminus \{\{a,b\}\}$

(c) $G \setminus \{\{a,b\}\}$

Fig. 6: Our graph G, and the subgraph $G \setminus \{\{a,b\}\}$.

Here are two theorems, both of which are true:

Theorem 2. *$G \setminus \{\{a,b\}\}$ is 3-colorable.*

Proof: A 3-coloring is shown in Fig. 6b. □

Theorem 3. *$G \setminus \{\{a,b\}\}$ is NOT 3-colorable.*

Proof: $G \setminus \{\{a,b\}\}$ is the complete graph K_4, as shown in Fig. 6c, and K_4 is not 3-colorable. □

Obviously, this is somewhat problematic. We might be tempted to claim that k-colorability is in fact either poorly defined or undecidable. However, complexity theorists have put a lot of work into proving that 3COL is NP-complete and thus in NP, which, by necessity, means that it is decidable.

In fact, simple extensions of Theorem 2 and Theorem 3 show that

$$\chi(G \setminus \{\{a,b\}\}) = 3 \quad \text{and} \quad \chi(G \setminus \{\{a,b\}\}) = 4,$$

where $\chi(G)$ is the chromatic number of G (i.e., the minimum number of colors needed to color G).

We are forced to conclude that $3 = 4$, and that in fact $0 = 1$.

B. *Adjacency*

One might be tempted to cut out the notion of coloring altogether, in the hope that the rest of graph theory is salvageable. Unfortunately, even the notion of adjacency yields contradiction. Let H be this graph:

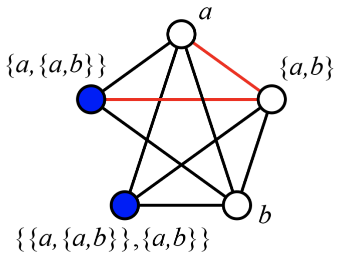


Fig. 7: Another perfectly well-behaved graph H.

Theorem 4. *In H, $\{\{a,\{a,b\}\},\{a,b\}\}$ is adjacent to $\{a,\{a,b\}\}$.*

Proof: $\{\{a,\{a,b\}\},\{a,b\}\} \cap \{a,\{a,b\}\} = \{\{a,b\}\}$, which is nonempty, so $\{\{a,\{a,b\}\},\{a,b\}\} \sim \{a,\{a,b\}\}$. □

Theorem 5. *In H, $\{\{a,\{a,b\}\},\{a,b\}\}$ is NOT adjacent to $\{a,\{a,b\}\}$.*

Proof: By the definition of H (as in Fig. 7), $\{\{a,\{a,b\}\},\{a,b\}\}$ is not adjacent to $\{a,\{a,b\}\}$. □

Thus, even adjacency, the most basic foundation of graph construction, leads to contradiction.

VI. Max-Overlap Labelings for Encryption

Maximum-overlap labelings also have great potential for use in encryption protocols.

Theorem 6. *A maximum-overlap labeling of a graph $G = (V, E)$ is cryptographically secure against all adversaries.*

Proof: Suppose that Alice wants to transmit graph $G = (V, E)$ to Bob. Alice computes the max-overlap labeling L of the graph and sends $(L(V), L(E))$, which Eve intercepts. One of two cases must occur:

- Eve attempts to read the input and falls asleep, or
- Eve concludes that Alice and Bob are insane, and that their transmitted graph must be worthless.

In both cases, Eve gains no information about G. □

VII. Poetry

A. The Limerick Graph

Let's take a look the max-overlap labeling of P_2, the length-2 path, as shown in 8:

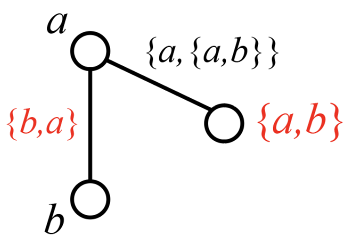


Fig. 8: Max-overlap labeling for P_2. As it turns out, this is the Limerick Graph $AABBA$.

The vertex and edge sets are

$$V = \{a, b, \{a, b\}\}$$

$$E = \{\{a, \{a, b\}\}, \{b, a\}\}$$

Verbalized, the edge set is “A, A, B, B, A”, or “AABBA.” This looks an awful lot like a rhyme scheme, doesn't it? Specifically, AABBA is the rhyme scheme for a limerick[5]. In general, it seems like we ought to be able to get graphs that correspond to other rhyme schemes.

B. The Sicilian Octave Graph

A Sicilian Octave is an eight-line poem with alternating rhymes; i.e., its rhyme scheme is ABABABAB. Consider this max-overlap labeling of the 3-clique K_3:

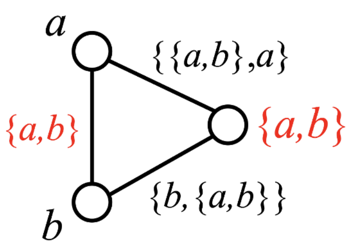


Fig. 9: Max-overlap labeling for K_3, the Sicilian Octave graph, with $\rho(G) = \frac{1}{3}$.

The edge set is

$$E = \{\{a, b\}, \{\{a, b\}, a\}, \{b, \{a, b\}\}\},$$

that is, ABABABAB.

C. The Shakespearean Sonnet Graphs (order 1 and 2)

We now explore several rhyme-scheme encoding graphs that may not necessarily be overlap-maximal, and which allow self-loops. This allows us to further bridge the (currently, very large) gap between mathematics and poetry.[6]

Shakespearean sonnets have a rhyme scheme of ABAB CDCD EFEF GG. Fig. 10 displays a graph with vertex and edge sets

$$V = \{a, b, c, d, e, f, g, \{a, b\}, \{c, d\}, \{e, f\}, \{g, g\}\}$$

$$E = \{\{a, b\}, \{\{a, b\}, \{c, d\}\}, \{c, d\}, \{e, f\}, \{\{e, f\}, \{g, g\}\}\}.$$ [7]

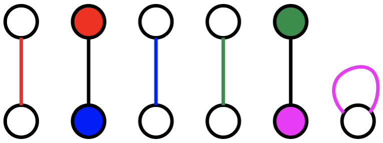


Fig. 10: The order-1 Shakespearean sonnet graph is a forest of K_2s, plus one self loop. It achieves $\rho(G) = \frac{2}{3}$.

This is a rather disappointing graph. Luckily for us, Shakespeare wrote more than one sonnet. Fig. 11 shows the graph corresponding to *two* Shakespearean sonnets. (Shakespeare wrote 153 sonnets; the order-153 Shakespearean sonnet graph is left as an exercise to the reader.)

[5] For example:

To label a 3-vertex path
For usage in improper math
Use $\{\{a, b\}, a, b\}$
On v_1 through $_3$;
Voila! It's a limerick graph.

[6] The two fields have been slowly growing apart since Omar Khayyam, although Lewis Carroll made a valiant effort to bring them back together.

[7] I know this overlaps the margin of the page. shhhhh.

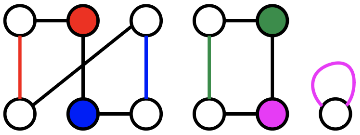


Fig. 11: The order-2 Shakespearean sonnet graph, with edges $E = \{\{a,b\},\{\{a,b\},\{c,d\}\},\{c,d\},\{e,f\},\{\{e,f\},\{g,g\}\},\{\{a,b\},a\},\{b,c\},\{d,\{c,d\}\},\{\{e,f\},e\},\{f,\{g,g\}\}\}$, i.e. ABAB CDCD EFEF GG ABAB CDCD EFEF GG. $\rho(G) = \frac{4}{11}$.

D. *Other Poems (Selected)*

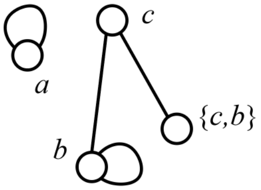


Fig. 12: Rhyme scheme graph of *The Raven* by Edgar Allen Poe, encoding rhyme scheme AABCCCBBB. $E = \{\{a,a\},\{b,c\},\{c,\{c,b\}\},\{b,b\}\}$ with $\rho(G) = \frac{1}{4}$.

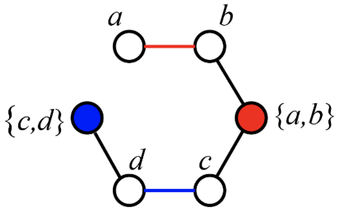


Fig. 13: Rhyme scheme graph of *The New Colossus* by Emma Lazarus (*"Give me your tired, your poor, your huddled masses yearning to breathe free"*), encoding rhyme scheme ABBAABBACDCDCD with $\rho(G) = \frac{2}{5}$.

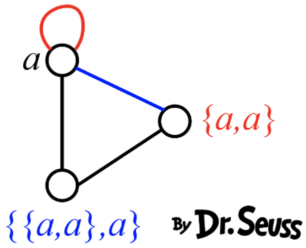


Fig. 14: Rhyme scheme graph of *Fox in Socks* by Dr. Seuss, encoding rhyme scheme AAAAAAAAAAAA. [8]

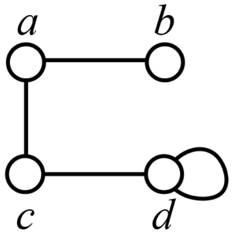


Fig. 15: Rhyme scheme graph of *Toxic* by Britney Spears, encoding rhyme scheme ABACDCDD with $\rho(G) = 0$.

VIII. Conclusion

In this work, we presented an overlap-maximization technique for labeling graphs. In order to counter the increasing ease of communication facilitated by the internet, we propose an alternate naming convention for graphs, in order to minimize clarity when spoken aloud.

In particular, scholars should strive to use either the name for a poem whose rhyme scheme encodes the graph, or directly list out the edge set of the maximal-overlap labeling of the graph.

For example:

1) the triangle graph could be validly called "AB, ABA, ABB" or "The Fox in Socks graph without self-loops";
2) the 4-cycle could be called "the second-smallest connected component of the Order 2 Shakespeare graph" or "AB, BAB, BABAB, ABAB."

Creativity is encouraged.

[8] not to be confused with the sound emitted by distressed humans

IX. References

[i] Google.

[ii] Wikipedia.

[iii] The Poetry Foundation.

All figures created in totally legal, non-pirated Adobe Illustrator.

X. Appendix A: Proof of Theorem 1

A. Upper bounding $\psi(G)$

First, we show that $\psi(G) \leq |V| - 2$.

Suppose for the sake of contradiction that $\psi(G) \geq |V| - 1$, i.e. that there are $|V| - 1$ vertices which share a label with an edge. Let these vertices be $V' = \{v_1, v_2, \cdots v_{|V|-1}\}$.

Note that these vertices are all of the form $\{x, y\}$ for some $x, y \in V$. So, $\subset$ is a strict partial order on the finite set V', so there exists at least one minimum element under $\subset$.

Let $m = \{m_1, m_2\}$ be such a minimum element. m_1 and m_2 must be vertices, since the vertex m overlaps with an edge. Since m is the minimum, $m_1, m_2 \notin V'$. So we have two vertices that are not in V', a contradiction. $\implies \impliedby$

Second, we show that $\psi(G) \leq |E|$. Note that

$$\begin{aligned}
\psi(G) &= \max_{L:V\to\mathbb{U}} |L(E) \cap L(V)| && \text{(Definition 1)}\\
&\leq \max_{L:V\to\mathbb{U}} |L(E)| \\
&= \max_{L:V\to\mathbb{U}} |E| && (L \text{ bijective})\\
&= |E|
\end{aligned}$$

as desired.

B. Constructively lower bounding $\psi(G)$

We show a recursive procedure to assign labels to a graph to ensure $\min(|E|, |V| - 2)$ overlap between edge and vertex labels.

In particular, it suffices to iteratively build up the graph by adding vertices and keeping the following invariant:

Invariant: A graph $G = (V, E)$ with $|V| \geq 2$ can be $\min(|E|, |V| - 2)$-overlap-labeled, with the labeling determined entirely by the labels of two "base" vertices (in particular, the two "base" vertices can be arbitrarily renamed, and the rest of the graph relabeled accordingly, while preserving the overlap).

Base cases: The theorem trivially holds for $|V| \in \{0, 1, 2\}$ [9].

For $|V| = 3$: The labelings displayed in Fig. 16 cover all cases. Note that a, b could be labeled anything, and the rest of the graph relabeled accordingly, while keeping the invariant true.

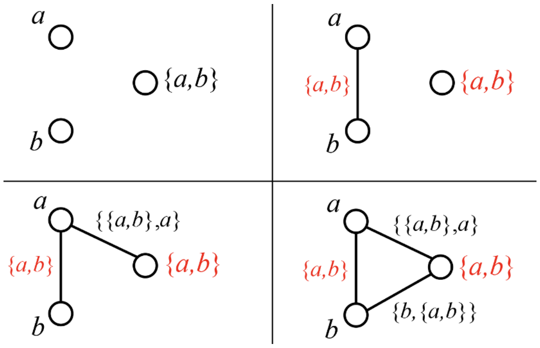


Fig. 16: Overlap-maximal labelings for $|V| = 3$, the base case for our recursive labeling scheme. Note that in each case the amount of overlap $\psi(G)$ is less than $\min(|E|, |V| - 2)$.

Inductive step. Consider some graph $G = (V, E)$ with $|V| \geq 4$. We split into cases:

- $|V| - 2 < |E|$. Then, there must be some vertex v with minimum degree. Inductively label $G \setminus v$, then choose some edge $e \in G \setminus V$ as the label for v.
- $|E| \leq |V| - 2$. Then, the graph is disconnected.
  - If one connected component is a singleton vertex, then label it arbitrarily, and inductively label the rest.
  - Otherwise, partition the graph into some connected component C and $G \setminus C$. Inductively label C and $G \setminus C$.

    Choose two edges $e_1, e_2 \in E(C) \setminus V(C)$ (i.e., they do not yet share a name with a vertex), and rename $G \setminus C$, using these e_1, e_2's labels as the "base" labels for $G \setminus C$.

It's easy to verify that this process preserves the invariant; the details are left to the reader.

[9] Whether $|V| = 0$ is a valid graph is up for debate; see "Is the null-graph a pointless concept?" by Harary and Read, 2006

XI. Appendix B: The Edge Set of K_7 Under Max-Overlap Relabeling

$$
\begin{array}{l}
E(K_7) = \{ \\
\quad \{a, b\}, \\
\quad \{a, \{a, b\}\}, \\
\quad \{a, \{a, \{a, b\}\}\}, \\
\quad \{a, \{\{a, \{a, b\}\}, \{a, b\}\}\}, \\
\quad \{a, \{b, \{\{a, \{a, b\}\}, \{a, b\}\}\}\}, \\
\quad \{a, \{\{\{b, \{\{a, \{a, b\}\}, \{a, b\}\}\}\}, \{a, b\}\}\}, \\
\quad \{b, \{a, b\}\}, \\
\quad \{b, \{a, \{a, b\}\}\}, \\
\quad \{b, \{\{a, \{a, b\}\}, \{a, b\}\}\}, \\
\quad \{b, \{b, \{\{a, \{a, b\}\}, \{a, b\}\}\}\}, \\
\quad \{b, \{\{\{b, \{\{a, \{a, b\}\}, \{a, b\}\}\}\}, \{a, b\}\}\}, \\
\quad \{\{a, b\}, \{a, \{a, b\}\}\}, \\
\quad \{\{a, b\}, \{\{a, \{a, b\}\}, \{a, b\}\}\}, \\
\quad \{\{a, b\}, \{b, \{\{a, \{a, b\}\}, \{a, b\}\}\}\}, \\
\quad \{\{a, b\}, \{\{\{b, \{\{a, \{a, b\}\}, \{a, b\}\}\}\}, \{a, b\}\}\}, \\
\quad \{\{a, \{a, b\}\}, \{\{a, \{a, b\}\}, \{a, b\}\}\}, \\
\quad \{\{a, \{a, b\}\}, \{b, \{\{a, \{a, b\}\}, \{a, b\}\}\}\}, \\
\quad \{\{a, \{a, b\}\}, \{\{\{b, \{\{a, \{a, b\}\}, \{a, b\}\}\}\}, \{a, b\}\}\}, \\
\quad \{\{\{a, \{a, b\}\}, \{a, b\}\}, \{b, \{\{a, \{a, b\}\}, \{a, b\}\}\}\}, \\
\quad \{\{\{a, \{a, b\}\}, \{a, b\}\}, \{\{\{b, \{\{a, \{a, b\}\}, \\
\qquad \{a, b\}\}\}\}, \{a, b\}\}\}, \\
\quad \{\{b, \{\{a, \{a, b\}\}, \{a, b\}\}\}, \\
\qquad \{\{\{b, \{\{a, \{a, b\}\}, \{a, b\}\}\}\}, \{a, b\}\}\} \\
\quad \}
\end{array}
$$

Neo-Classical Logic

Une Logique non classique[*]

Martin Vassor Fangyi Zhou

Resume.

Standard logics, such as *classical logic* or *intuitionist logic* suffer from flaws that remained unaddressed as of today (like philosophical questions about using the axiom of *excluded middle*, which draws the line between the two logic mentioned previously). In this paper, we introduce *neo-classical logic*, in an attempt to reconciles classical and intuitionistic logics, as well as to capture some real-world event which can not be explained by previous logics, due to their lack of expressivity.

After introducing its syntax and semantique, we study the meta-theory of this new logic, and show multiple interesting results, such as the *law of contradiction*, or that validity is decidable, which states that a proposition can simultaneously hold and not hold.

1 Introduction

Motivation. Our paper is motivated by strong empirical evidences that some events, concepts, objects can simultaneously exist and don't exist (see Figure 1, but also [6, 8, 10]).

Despite strong evidence, to the best of our knowledge, no logic seems to be expressive enough to prove statements as simple as « Exists *a* such that *a* does not exist. » This absence (or possibly presence) shows that there exists a gap between current theoretical models and the actual reality.

Contribution. Our main contribution in this paper is to introduce a new logic, *neoclassical logic*[1], which reduces this gap, allowing us to express more faithfully the subtleties of the world.

Moreover, we demonstrate how we use neo-classical logic to reconcile two important logic theories, namely classical logic and intuitionistic logic, with a *constructive* proof of the *Law of Excluded Middle* in our neo-classical logic. Neo-classical logic subsumes classical logic and intuitionistic logic.

Outline. In Sections 2 et 3, we discuss the limitations of the two most popular logics, namely classical and intuitionistic logic. From that, we conclude that there is a need to fill the gap left opened by those two logic systems; in order to capture real-life cases that can not be addressed by the two logic aforementioned. In Section 4, we introduce our proposal, which we call *neo-classical logic*, by successively presenting both its syntax and its semantique. In Section 5, we discuss the meta-theory of the logic. Finally we present an actual use case in Section 6, showing that our logic allows to capture complex arguments while remaining decidable. Finally, in Section 7, we conclude by discussing some limitations, presenting some unrelated work and highlight some interesting research paths for future work.

2 Classical Logic is Not Modern

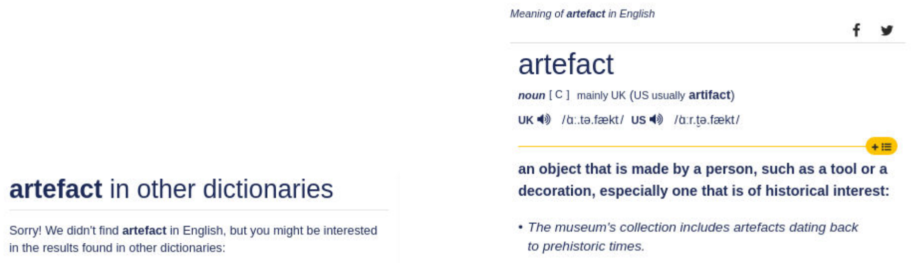
We look up the dictionary entry of the word « classical » in Cambridge dictionary again [9], and see what it could mean. The main usages include « traditional in style or form, or based on methods developed over a long period of time, and considered to be of lasting value », and « used to describe something that is attractive because it has a simple, traditional style. »

We also note that this sentence was provided as an exemple:

> Does she study classical ballet or modern ballet?

[*] A paper on logic must have some French, but we don't know how to typeset accents, so here it is.

[1] Also spelled « neo-classical » due to our lack of motivation to check the consistency across the paper.



artefact in other dictionaries

Sorry! We didn't find **artefact** in English, but you might be interested in the results found in other dictionaries:

Meaning of ***artefact*** *in English*

artefact

noun [C] mainly UK (US usually **artifact**)

UK /ˈɑː.tə.fækt/ **US** /ˈɑːr.t̬ə.fækt/

an object that is made by a person, such as a tool or a decoration, especially one that is of historical interest:

- *The museum's collection includes artefacts dating back to prehistoric times.*

(a) Evidence of the non existence of the word « Artefact » [10]

(b) Evidence of existence of the word « Artefact » [8]

Figure 1: Empirical evidence of the simultaneous existence and non-existence of the word « Artefact » in English, according to the Cambridge Dictionary.

This sentence gives a strong hint that *classical* ballet is *not modern*. We are also confident to conclude that classical logic is not modern, via a simple application of substitution. A modern-day logic should be modern, in order to catch up with contemporary development of real life events and technological advancements. Moreover, it is an unfortunate fact that classical logic cannot represent the simultaneous existence and non-existence of a word (cf. Figure 1) — it cannot model contemporary dictionaries in a modern world.

3 Intuitionistic Logic is Not Intuitive

In the first courses of computer science, students learn classical logic due to its simplicity. Some mathematicians and theoretical computer scientists, however, prefer intuitionistic logic.

Unfortunately, we are unable to find an entry for « intuitionistic » in Cambridge dictionary, which is a strong hint that it is not intuitive. Think about that, why would computer scientists call something intuitive using a complicated word that looks like « intuitive », but decide against its use?

4 Neo-Classical Logic

In order to address the unsatisfactory deficiency of both classical logic and intuitionistic logic, we introduce *neo-classical logic*.

4.1 Formules

$$\begin{array}{llll}
\mathcal{A} & ::= & a, b, c, \ldots & \textit{Atomic propositions} \\
\mathcal{P}, \mathcal{Q} & ::= & \mathcal{A} & \textit{Atomic proposition} \\
 & \mid & \neg\mathcal{P} & \textit{Negation of P} \\
 & \mid & \mathcal{P} \wedge \mathcal{Q} & \textit{And}
\end{array}$$

Notice that, together with the semantique we define below, and in particular with the traditional encoding of $P \vee Q$ as $\neg(\neg P \ \wedge \neg Q)$ and $P \Rightarrow Q$ as $\neg(P \wedge \neg Q)$, we can have usual constructs. Therefore, we allow ourself to use those elements as needed.

The syntax of formules does not differ much from classical or intuitionistic logic — this is why we think our logic is neo-classical: it looks classical, but is quite modern. We explain the modernness in detail when we discuss the semantique.

We fix a set $\mathcal{A}$ of atomic propositions, ranging over $a, b, c, \ldots$. Atomic propositions form the basic form of neo-classical logic formules, and we also recognise standard logical operators: negation, conjunction, disjunction, and implication.

4.2 Semantique

Let $\vdash$ be a (binary) predicate defined according to the rules in Figure 2. We use an infix notation: $\Gamma \vdash P$ where Γ is an environment and P a formula of the neo-classical logic. We write $\Gamma \nvdash P$ for $\neg(\Gamma \vdash P)$.

$$\frac{a \notin \Gamma}{\Gamma \vdash \neg a}\ (\neg\text{Exists}) \qquad \frac{a \in \Gamma}{\Gamma \vdash a}\ (\text{Exists}) \qquad \frac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1 \cup \Gamma_2 \vdash P \wedge Q}\ (\wedge) \qquad \frac{\Gamma \vdash \neg P}{\Gamma \vdash \neg(P \wedge Q)}\ (\text{DeMorganL})$$

$$\frac{\Gamma \vdash \neg Q}{\Gamma \vdash \neg(P \wedge Q)}\ (\text{DeMorganR}) \qquad \frac{\Gamma \vdash P}{\Gamma \vdash \neg\neg P}\ (\neg\neg\mathrm{I})$$

Figure 2: Inference rules of the $\vdash$ predicate

The novelty of neo-classical logic lies in the rules ($\neg$Exists) and (Exists). Those two rules establish a strong correspondence between a witness (or observation) of an event and establishing the existence of that event. On the one hand, rule ($\neg$Exists) states that if we have no evidence of an event, we can conclude that this event does not exist. This is, informally, motivated by the fact that having no evidence of an event is indistinguishable from that event not existing. Indeed, if there is a way to distinguish two possible worlds, one with the event, one without; then the distinctive element actually serves as witness of the event. Therefore, from Occam's razor (often stated « Pluralitas non est ponenda sine necessitate »[2], and widely spread in the literature, see e.g. [7, Book I, 4, 188a17], but also [4, Prima Pars, Q.2 art.3 -AG2, &c.]), we should not suppose the existence of an event we have no evidence of; which ends the justification of that rule.

Conversely for (Exists), if we have an observation of an event a, we can deduce the existence of that event. This is trivially motivated.

Exemple 1 (Artefact exists and does not exist).

$$\frac{\dfrac{}{\text{artefact} \vdash \text{artefact}}\ (\text{Exists}) \qquad \dfrac{}{\varnothing \vdash \neg\text{artefact}}\ (\neg\text{Exists})}{\text{artefact} \vdash \text{artefact} \wedge \neg\text{artefact}}\ (\wedge)$$

Theoreme 2 (Consistency). *The deduction system is consistent, i.e. $\varnothing \nvdash \bot$.*

Preuve. It is not possible to introduce $\bot$. Voilà.

Remarque 3 (Absence of Absolute Truths And Absolute Falsities). We note that $\top$ and $\bot$ are not formules of neo-classical logic. This is important since there are no absolute truths and absolute falsities in the modern world. We make a conscious effort to model this observation in our logic.

5 Meta[3]-Theory of *Neo-Classical Logic*

Lemme 4 (To tree or not to tree).

$$\forall\Gamma \,.\, \forall P \,.\, \Gamma \vdash P \vee \Gamma \nvdash P$$

Preuve. First, we need to clarify the statement. It is obviously not a neo-classical formula (typically, neo-classical logic do not have quantifier). Instead, it is a formula from first-order classical logic, where $\vdash$ is a (binary) predicate; and $\vee$ is the disjunction of classical logic.

The result then follows directly from the law of excluded-middle of classical logic. Voilà.

Lemme 5 (If no proof, then proof of negation).

$$\forall\Gamma \,.\, \forall P \,.\, \Gamma \nvdash P \Rightarrow \Gamma \vdash \neg P$$

Preuve. By induction on P.

Case $P = a$: from the premisses of (Exists), we have that $a \notin \Gamma$. Therefore, rule ($\neg$Exists) applies.

Case $P = \neg Q$: neither ($\neg$Exists), (DeMorganL), (DeMorganR) nor ($\neg\neg$I) apply. By case analysis on Q:

[2] « entities should not be multiplied beyond necessity », translation Wikipedia.

[3] Not to be confused with Meta Platforms Inc, "metaverse", or whatever.

Case Q **is** a: since ($\neg$Exists) does not apply, we have that $a \in \Gamma$, therefore $\Gamma \vdash Q$, therefore $\Gamma \vdash \neg P$ by applying ($\neg\neg$I).

Case Q **is** $\neg Q'$: since ($\neg\neg$I) does not apply, we have that $\Gamma \nvdash Q'$. From the induction hypothesis, we have that $\Gamma \vdash \neg Q'$. Therefore, $\Gamma \vdash \neg P$ (i.e. $\Gamma \vdash \neg\neg\neg Q'$) by applying ($\neg\neg$I).

Case Q **is** $Q_1 \wedge Q_2$: since neither (DeMorganL) nor (DeMorganR) apply, we have that both $\Gamma \nvdash \neg Q_1$ and $\Gamma \nvdash \neg Q_2$. Therefore, from the induction hypothesis, both $\Gamma \vdash \neg\neg Q_1$ and $\Gamma \vdash \neg\neg Q_2$. From the premises of ($\neg\neg$I) on those two statements, we have that both $\Gamma \vdash Q_1$ and $\Gamma \vdash Q_2$. Therefore, $\Gamma \vdash Q_1 \wedge Q_2$ (i.e. $\Gamma \vdash Q$), by applying ($\wedge$). Therefore, we can prove $\Gamma \vdash \neg P$, i.e. $\Gamma \vdash \neg\neg Q$, by applying ($\neg\neg$I).

Case $P = Q_1 \wedge Q_2$: Since rule ($\wedge$) does not apply, we have that for any subsets Γ_1, Γ_2 of Γ, neither $\Gamma_1 \vdash Q_1$ nor $\Gamma_2 \vdash Q_2$. I.e., $\forall \Gamma' \in \Gamma \,.\, \Gamma \nvdash Q_1$ (resp. Q_2). Therefore, from the induction hypothesis, we have that for any subset Γ' of Γ, $\Gamma' \vdash \neg Q_1$ (resp. Q_2).

Therefore, the proof finishes by showing that $\Gamma \vdash \neg P$, i.e. $\Gamma \vdash \neg Q_1 \wedge Q_2$ by applying either (DeMorganL) or (DeMorganR). Voilà.

5.1 Law of Contradiction

Lemme 6 (Empiricism brings knowledge).

$$\forall P \,.\, \exists \Gamma \,.\, \Gamma \vdash P$$

Preuve. By induction on P.

Case $P = a$: let $\Gamma = \{a\}$, apply (Exists).

Case $P = \neg P'$: By case analysis on P':

Case $P' = a$: Let $\Gamma = \varnothing$, and the result holds directly from rule ($\neg$Exists).

Case $P' = Q_1 \;\wedge Q_2$: From the induction hypothesis, we know that there exist Γ_1 such that $\neg Q_1$ holds. Therefore, we can take $\Gamma = \Gamma_1$ and apply (DeMorganL).

Case $P' = \neg Q$: In that case, we have to prove $\neg\neg Q$. The result holds directly from the induction hypothesis and by applying the rule ($\neg\neg$I).

Case $P = Q_1 \wedge Q_2$: From the induction hypothesis, we know that there exists a Γ_1 (resp. Γ_2) such that $\Gamma_1 \vdash Q_1$ (resp. $\Gamma_2 \vdash Q_2$). We can take $\Gamma = \Gamma_1 \cup \Gamma_2$ and apply the rule ($\wedge$). Voilà.

Theoreme 7 (Law of contradiction).

$$\forall P \,.\, \exists \Gamma \,.\, \Gamma \vdash P \wedge \neg P$$

Preuve. This Theoreme is a corollaire of lemme 6. Voilà.

5.2 Excluded-Middle

Theoreme 8 (Excluded-Middle).

$$\forall \Gamma \,.\, \forall P \,.\, \Gamma \vdash P \vee \neg P$$

Preuve. As we said above, $P \vee \neg P$ is encoded in our calculus as $\neg(\neg P \wedge \neg\neg P)$. Therefore, we actually have to prove that this formula holds for all P and Γ.

From lemme 4, $\Gamma \vdash P$ or $\Gamma \nvdash P$. By case analysis:

Case $\Gamma \vdash P$:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash P}\,\text{Hypothesis}}{\Gamma \vdash \neg\neg P}\,(\neg\neg\mathrm{I})}{\Gamma \vdash \neg(\neg P \wedge \neg\neg P)}\,(\text{DeMorganL})$$

Case $\Gamma \nvdash P$**:** From lemme 5, $\Gamma \vdash \neg P$.

$$\cfrac{\cfrac{\cfrac{\vdots}{\Gamma \vdash \neg P}\ \text{Hypothesis}}{\Gamma \vdash \neg\neg\neg P}\ (\neg\neg\mathrm{I})}{\Gamma \vdash \neg(\neg P \wedge \neg\neg P)}\ (\text{DeMorganR})$$

Voilà.

5.3 Decidability of Validity

Finally, we want to show that deciding whether a formula P is valid under a context Γ is decidable. For that, we first show how to compute two sets: first, the set of event that *must* be in Γ, and second, the set of event that must not be in Γ, in order to satisfy the formula. Finally, we compare Γ with those two sets to decide whether P can be satisfied.

Set of required events. Let $\mathbb{R}_P$ be the set of required events to satisfy P. Notice that there might be multiple ways to satisfy a single formula, i.e. multiple sets of required events are possible. Therefore, $\mathbb{R}_P$ is actually a set of sets of events. $\mathbb{R}_P$ can easily be defined as follows:

$$\mathbb{R}_P \overset{\text{def}}{=} \begin{cases} \emptyset & \text{if } P = \neg a, \forall a \\ \{\{a\}\} & \text{if } P = a, \forall a \\ \{\mathbb{P}_{Q_1} \cup \mathbb{P}_{Q_2} | \mathbb{P}_{Q_1} \in \mathbb{R}_{Q_1} \wedge \mathbb{P}_{Q_2} \in \mathbb{R}_{Q_2}\} & \text{if } P = Q_1 \wedge Q_2 \\ \mathbb{R}_{\neg Q_1} \cup \mathbb{R}_{\neg Q_2} & \text{if } P = \neg(Q_1 \wedge Q_2) \\ \mathbb{R}_Q & \text{if } P = \neg\neg Q \end{cases}$$

Lemme 9 (Required is correct)**.** *For all Γ, for all P, if $\nexists \mathbb{P} \in \mathbb{R}_P$ such that $\mathbb{P} \subseteq \Gamma$, then $\Gamma \nvdash P$.*

Preuve. The proof is direct, by induction on P. Each possible case of corresponds to a case of $\mathbb{R}_P$, corresponds to the premises of a rule (or two rules for the two variants of (DeMorgan)). Voilà.

Lemme 10. *For all P, $\mathbb{R}_P$ is computable.*

Preuve. Let $n_1(P)$ be the number of occurrences of $\neg(Q_1 \wedge Q_2)$ in P. Let $n_2(P)$ be the number of occurrences of $\neg Q$ in P. Let $\prec$ be an order relation on formules defined as follows:

$$P \prec Q \text{ if and only if } (n_1(P) < n_1(Q)) \text{ or } (n_1(P) = n_1(Q) \text{ and } n_2(P) < n_2(P))$$

We easily show, by induction, that, for each recursive computation of $\mathbb{R}_Q$ to compute $\mathbb{R}_P$, $Q \prec P$. Also, there is a finite number of recursive call at each step.

Therefore, the computation eventually terminates. Voilà.

Set of forbidden events. Let $\mathbb{F}_P$ be the set of forbidden events to satisfy P. $\mathbb{F}_P$ can easily be defined as follows:

$$\mathbb{F}_P \overset{\text{def}}{=} \begin{cases} \{a\} & \text{if } P = \neg a, \forall a \\ \emptyset & \text{if } P = a, \forall a \\ \mathbb{F}_{Q_1} \cap \mathbb{F}_{Q_2} & \text{if } P = Q_1 \wedge Q_2 \\ \mathbb{F}_{\neg Q_1} \cup \mathbb{F}_{\neg Q_2} & \text{if } P = \neg Q_1 \wedge Q_2 \\ \mathbb{F}_Q & \text{if } P = \neg\neg Q \end{cases}$$

Lemme 11 (Forbidden is correct)**.** *For all Γ, for all P, if $\Gamma \cap \mathbb{F}_P \neq \emptyset$, then $\Gamma \nvdash P$.*

Preuve. The proof is direct by induction on P. Each possible case of corresponds to a case of $\mathbb{F}_P$, corresponds to the premises of a rule (or two rules for the two variants of (DeMorgan)). Voilà.

Lemme 12. *For all P, $\mathbb{F}_P$ is computable.*

Preuve. The proof is similar to the proof of lemme 10. Voilà.

Theoreme 13. *For all* Γ*, for all* P*,* $\Gamma \vdash P$ *if and only if:*

1. $\exists \mathbb{P} \in \mathbb{R}_P \,.\, \mathbb{P} \subseteq \Gamma$*; and*
2. $\Gamma \cap \mathbb{F}_P = \emptyset$*.*

Preuve. The if direction is a direct consequence of lemmes 9 et 11, by contraposition.

We have to show the other direction, that is the two conditions are sufficient to have $\Gamma \vdash P$.

The result is direct by induction on P. Voilà.

Corollaire 14 (The validity of a formula is decidable)**.** *For any* Γ*, for any* P*,* $\Gamma \vdash P$ *is decidable.*

6 Practical Applications

As logicians, we have a moral duty to remain close to practical applications of our works. Indeed, the formal study of arguments allows every citizen to cast a light and understand actual facts on the world that surrounds us. In order to show that our work has indeed some practical use, in this section, we show an actual exemple taken from everyday's life.

While police violence is a well-documented issue in modern democracies [1,5], we still have statements that police violence does not exist [2,3] (e.g. « Ne me parlez pas de [. . .] violences policières, ces mots sont inacceptables dans un État de droit. »[4], Macron *et al.*, reported at 0:40 in [3], and in [2]).

In the following, we show that our logic captures such arguments, as they are actually derivable. Let v_p an actual observation of police violence. In the following, we show how we can simultaneously claim that such event occurs and does not occurs, given such observation.

$$\dfrac{\dfrac{\dfrac{}{v_p \in \{v_p\}}\,\text{Set theory}}{\{v_p\} \vdash v_p}\,\text{(Exists)} \qquad \dfrac{\dfrac{}{v_p \notin \varnothing}\,\text{Set theory}}{\emptyset \vdash \neg v_p}\,(\neg\text{Exists})}{\{v_p\} \vdash v_p \wedge \neg v_p}\,(\wedge)$$

7 Conclusion, Unrelated and Future Work

In this paper, we presented new semantique for the propositional calculus. We show that this new semantique have interesting properties. For instance, the standard law of excluded-middle holds in our calculus, or that the validity of a formula is decidable. In addition, we show that new, previously unexplored[5] laws, such as the *Law of contradiction* also holds for our calculus.

Overall, as stated in the motivation, our calculus captures intuitive notions of proofs based on the notion of evidence. Our calculus finally brings the formal basis needed for reasoning on the ontological[6] notion of existence, which is a problem that was left opened since almost 2.5 millennia; yet still relevant as of today.

Unrelated Work. Very little is unrelated to this one, as this work regards the relation between evidence and knowledge. Therefore, all evidence-based sciences are related to this work. Furthermore, as it aims to give a formal basis to ontological[7] arguments, it relates to philosophical works, which we therefore have to exclude from unrelated works as well.

Future Work. The current main limitation is that it is based on propositional logic; and therefore lacks the expressiveness needed to capture more complex argumentations. Future work could include extending this paper up to higher-order logic. Such extension could benefit from an extended expressiveness, possibly being expressive enough to encode arithmetics.

The exemple shown in Section 6 illustrate that our logic allows a fine characterisation of the reasoning that leads to some claims. For instance, we saw that the authors of the sentence above (Macron *et al.*, reported in [2,3]) loose some information in their reasoning. The current work is limited in that it does not give explanation for this loss. Intuitively, we can hypothesize various causes, ranging from plain lack of knowledge to deliberate ignorance. Future works is needed to elaborate on such hypothesis.

[4] « Don't say [. . .] "police violence" to me, those words are unacceptable in a Rechtsstaat. » Translation by the authors.

[5] To the best of the author's knowledge.

[6] Ditto.

[7] or « epistemic », as you can guess, the authors are not philosophers and are just using those big words to impress reviewer #2.

Remerciements

We thank anonymous reviewers from Sigbovique 2022 for their helpful comments and insights, except reviewer #2. We think this paper should be rated 5/7.

References

[1] Violences policières, les situer pour mieux y résister. *Vacarme 77*, 4 (2016), 8–23. Place: Paris Publisher: Association Vacarme.

[2] VIDEO. "Gilets jaunes" : Macron juge "inacceptable dans un Etat de droit" de parler de "violences policières", Mar. 2019.

[3] Clique TV. Clément Viktorovitch : Peut-on parler de violences policières ? - Clique - CANAL+, June 2020.

[4] d'Aquino, T. *Summa Theologiae.* XIII AD.

[5] Jobard, F. *Bavures policières ? La force publique et ses usages*. TAP / Politique et société. La Découverte, Paris, 2002.

[6] Schrödinger, E. Die gegenwärtige Situation in der Quantenmechanik. *Naturwissenschaften 23*, 48 (Nov. 1935), 807–812.

[7] Ἀριστοτέλης. Φυσικὴ ἀκρόασις. c. IV BC.

[8] University of Cambridge. Cambridge dictionary – Entry "Artefact", 2022.

[9] University of Cambridge. Cambridge dictionary – Entry "Classical", 2022.

[10] University of Cambridge. Cambridge dictionary – Spellcheck of "Artefact", 2022.

A Patriotic Analysis of Programming Paradigms

Jacob Weiner
Abraham Lincoln Chair for Patriotism
Founding Fathers University

SIGBOVIK 2022

Abstract

In this paper, we will examine why dependent type theory is the most patriotic paradigm for programming. We will examine several key founding documents to draw our conclusion. We will view many American flag images and feel patriotic.

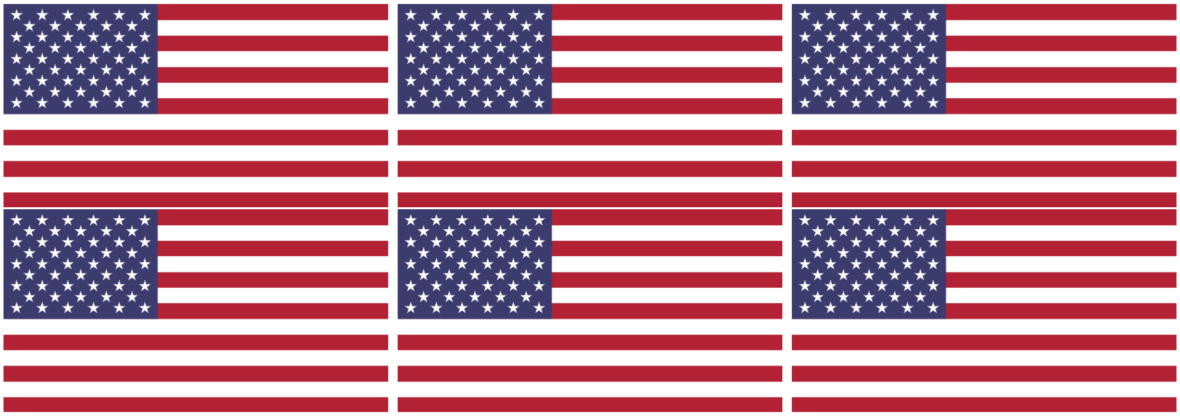
1 Introduction

It is universally accepted among Americans that patriotism is the highest and most important virtue[1]. Hence, it is critical to the overall well-being of our nation and the people within it to identify what paradigms and techniques in computer science are deserving of our glorious star-spangled flag, and which are being secretly promulgated by evil enemy spies. Thus, we can rid ourselves of those elements which hold us back from the glorious future which our founders intended for us.

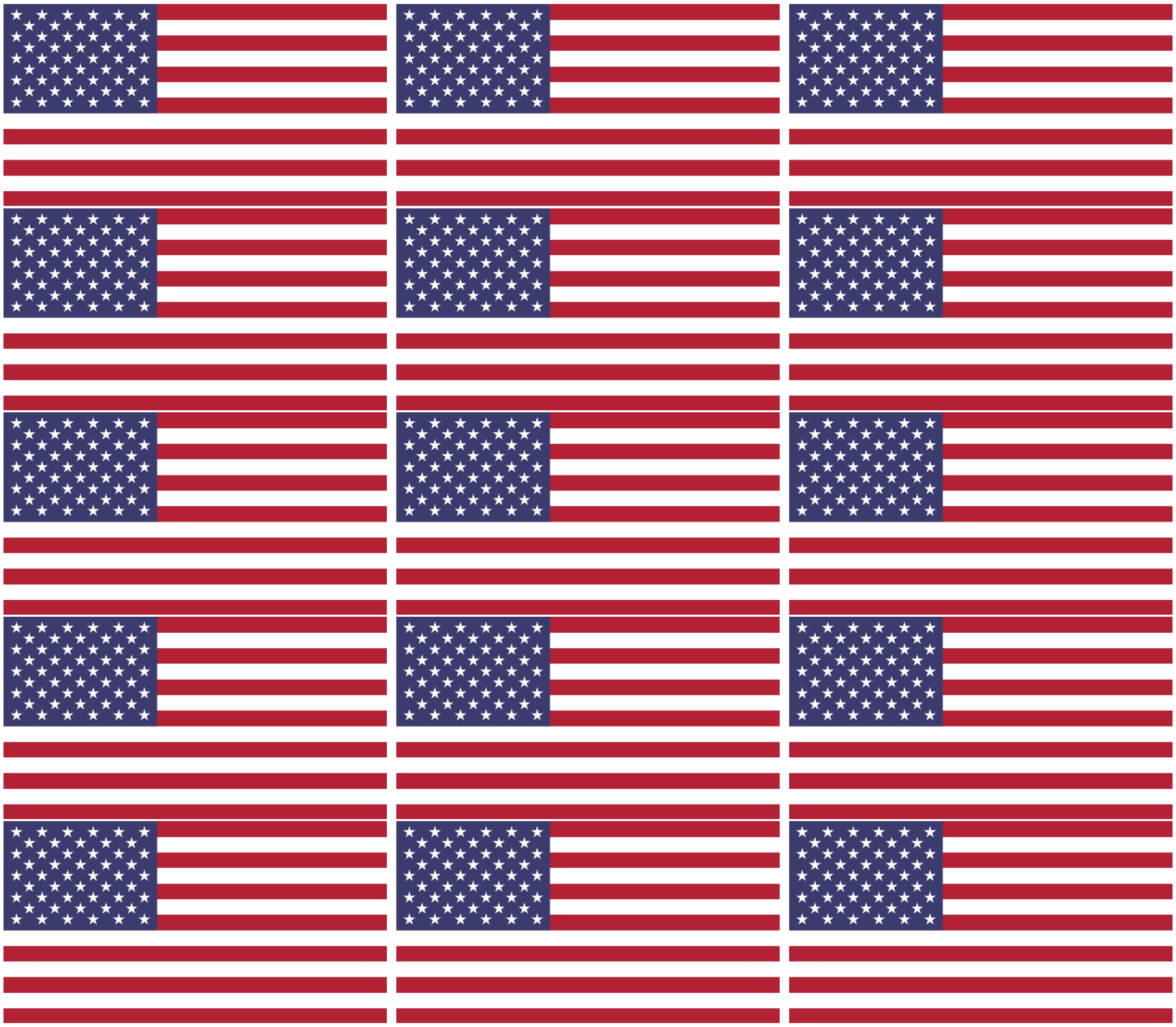
To quote John Winthrop, "...we shall be as a [shining] city upon a hill" if all our code is truly *American*.

Figure 1: AMERICA

2 Motivation



[1] If you dissent, you are an **unpatriotic traitor!!!!!!!**



My coding 'tis of thee
Sweet land of liberty
Of thee I sing...

Land where the simply typed
Lambda calculus thrives,
From every shell, a cry:
"Let FP ring!"

3 Methods

I examined founding documents and common sayings for out-of-context quotations to support my thesis.

I used American flags to distract from several questionable logical leaps.

Figure 2: AMERICA

4 Analysis of Data and Results

The first hint on our quest to find the most patriotic paradigm is in our core document, the Consitution. The Constitution defines what it means to be American[2], and no part of it more so than the Preamble.

The Preamble states that the goal of America is to form a more perfect **union**. Therefore, any paradigm which hopes to be patriotic must support union types. This allows for C, which features `union` types, as well as functional languages, which allow for datatypes with multiple constructors.

The importance of union types can be seen in Article 1, Section 1. This section clearly describe "member of congress" a union type: either a `Senator` or a `Representative`.

Figure 3: AMERICA

If any document hopes to come close to the Constitution in patrioticity[3], it is the Declaration of

[2] Obviously.

[3] Yes, I just created that word.

Independence, crafted by the great Thomas Jefferson. It notes that all human beings have certain "unalienable rights".

Note the importance of the word **unalienable** – it means that these rights can neither be altered nor removed. Hence, we ***must***[4] conclude that immutability of data is a core patriotic value, without which our country would fall to ruins.

Only one paradigm offers immutability, along with the union types so highly prized by the Constitution. This is **functional programming**.

One curious note about the Constitution is that Congress is described in article 1, while the Presidency is delayed until article 2. This is because the founders believed that deliberation is more important than action. By extension$_{\text{ality}}$, we must conclude that they would regard imperative programming to be inferior, as it is a paradigm of action.

Figure 4: AMERICA

So we are now absolutely sure that the paradigm of our founding fathers is functional. But what flavor of functional programming? Another look at the Declaration of Independence will illuminate us to the true intentions of Washington, Franklin, Jefferson, and all the other great ones.

"We hold these *truths* to be self-*evident*: that all men are created *equal*..."

Note the three key words in this patriotic phrase:

- **Truth** and **evident** indicate that we ought to have a way to create evidence that our programs are working correctly.

- The mention of **equality** indicates that all types should be equality types.

The latter condition is rather stringent, but it can be done if we allow for function extensionality. But to use function extensionality, we need to express propositions with universal quantifiers within our language! How can this be done?

There is only one answer: we must use dependent type theory with the Curry-Howard Correspondence[5].

In fact, we could have seen this from the beginning, for it is often said that there is nothing as American as apple pie – and dependent type theory includes at its core Pi-types!

One piece of further evidence that dependent type theory is what our founders would have supported comes from the 9th amendment in the Bill of Rights. It notes that the enumeration of rights "should not be construed" to imply that there were not further rights. This clearly implies

[4] At least, those of us who are truly AMERICAN.

[5] Both Curry and Howard are good old American citizens.

Figure 5: AMERICA

that the set of all rights to which American citizens are entitled is in fact an uncountable set, as they cannot be enumerated!

Dependent type theory easily handles creating and reasoning about uncountable types, while few other paradigms truly support uncountability – even so-called "real number types" in many languages, from C to SML, are limited by machine precision. Yet using dependent type theory, one must represent real numbers precisely – or else not at all!

Figure 6: AMERICA

5 Conclusions

Dependent type theory should be the only paradigm used by true American computer scientists and programmers. Any other paradigm would cause the founding fathers to roll over in their graves.

6 Recommendations

Ban all other paradigms from American soil immediately.

7 Future Research

This highly innovative method can be used to evaluate differing techniques in many other areas of human effort and ingenuity, including science, mathematics, art, literature, social sciences, business, basket weaving, red herring catching, and WiFi customer service.

8 Impact

America will be better because of this research. I am sure of it.

Figure 7: AMERICA

9 Acknowledgments

George Washington is to be acknowledged.

Figure 8: AMERICA

On Ruinment
Ruination Theory and its Consequents

Luna A. تاشتر (they/them)[0]
Dissociation for Heresiographical Informatics
disinformatics@pm.me

***Abstract*—TODO: write abstract**

I. Introduction

The field of Ruination Theory is a severely sparse and understudied field, despite its deep ramifications. The question of ruinment is fundamental to our very perception of the world and the categorical[1] distinctions we make of it. We seek to demonstrate the importance of Ruination Theory through our novel Waldstreicher–Equivalence Theorem, deriving results impossible to find in any other field of Computer Science or Mathematics, as well as fill in the as of yet missing fundamentals to this incredibly impactful and meaningful field.

II. Ruinment

We refer to the act of ruining something as *ruinment*, [2] and begin with the central lemma of our study, which is perhaps the most important fact of ruinment.

Lemma II.1 (Tascheter–Conover Fundamental Lemma of Ruinment)**.** *Adam ruins everything.*

Proof. See [3]. □

In search of a mathematical formalization of the notion of ruinment, we assert the following characteristics that must be true for such a formalization:

Proposition II.2. *Anything can ruin anything else.*

Proof. Take two objects A and B which may be enumerated within a given language of discourse.[3] We call A the ruinator and B the ruined. Note that the phrase "A ruins B" is semantically valid. We exemplify three cases: "*Fire* ruins *the Library of Alexandria*"[2], "*Testosterone* ruins *Luna A. تاشتر* "[1], and "*Adam* ruins *everything*" (by the Tascheter–Conover Fundamental Lemma of Ruinment). □

Exercise 1. *Translate this paper into a non-English language.* [4]

Proposition II.3. *The classes of all ruinators and ruineds are finite sets.*

Proof. First, note that language, as a general construct, exists within humans for the purpose of communication. Thus, the class of all words in every language of discourse must fit within the combined storage space of the minds of every human. As there is a finite amount of humans, [5] each human stores information in a physically-bounded region called the Brain[4], and any given region of space has an upper limit to information density[5], this combined storage space must be finite. Thus, the class of all words in every language of discourse is finite, and thus we may form a set of all words in every language of discourse $\mathfrak{L}$. As the class of all English words $\mathfrak{E} \subset \mathfrak{L}$, $\mathfrak{E}$ is finite and thus also forms a set. We note that each locus[6] in English is a finite sequence of words. As every ruinator and ruined corresponds to a locus, there are finite ruinators and ruins. Thus, their classes are finite sets. □

Proposition II.4. *The sets of ruinators and ruineds are equal.*

Proof. Trivially, by Proposition II.2.[7] □

Proposition II.5. *Reflexivity of ruinment.*

Proof. Everybody eventually ruins themself[7]. We extend this notion to apply to *everything* as well: write $\mathfrak{R}$ as the set of ruinators and ruineds. Then, $\forall r \in \mathfrak{R}$, create some embodiment $E(r)$ as follows: if r is a person, $E(r) = r$, and otherwise construct $E(r) = T(S(I(r)))$ where $I(r)$ is the platonic ideal of r, $S(x)$ bestows sentience upon x, and $T(x)$ turns x into a hot Tumblr Sexyman.[8] We note that this process has already been proved feasable on Van Gogh's *The Starry Night* [8], and, in fact, may be undergone in general[9].

$$\begin{tikzcd}
\text{Person} \arrow[loop left, "E"] \arrow[rr, dashed, leftrightarrow, "E"'] & & \text{Thing} \arrow[d, "I"] \\
\text{Sexyman} \arrow[u, "\subset"] & \text{Sentient} \arrow[l, "T"] & \text{Ideal} \arrow[l, "S"]
\end{tikzcd}$$

[0] تاشتر is transliterated *Tascheter*, for those of you who have spoken solely the profane Latin-scripted tongues.

[1] In a Hegelian sense, you dirty Category[13] Theorists.

[2] Other terms used in the field include *ruination*, *T0T41 PWN4G∃*, and *ratio*.

[3] We assume the use of English in this paper due to the author not knowing any other language. Je sais un peu de français, mais c'est une langue merdeuse, donc je ne veux pas le utiliser.

[4] Get it published in SIGBOVIK 2023 and I'll mail you 1 USD! Please, I need citations.

[5] ٧٬٩٣٥٬٨٨٤٬٢٥٦[6]

[6] A phrase used to uniquely refer to an object.

[7] En français, je peux dire tout ce que je veux. La réalité est une illusion! Le univers, c'est un hologramme! Achète l'or!

[8] Pardon mon anglais.

As $T \circ S \circ I$ is an homomorphism over ruination,[9] everything eventually ruins itself. □

Proposition II.6. *Transitivity of ruinment.*

Proof. Suppose there are $A, B, C \in \mathfrak{R}$ where A ruins B and B ruins C. We prove, then, that A must ruin C. We propose a well-studied mechanism: the Sandler–Hallmark Theorem. In short, this theorem generalizes the mechanism through which movies are still well-liked[10]. We note that Adam Sandler and Hallmark both ruin movies; [10] however, movies aren't ruined yet[10]. The discoverers of the Sandler–Hallmark Theorem, in their seminal [11] paper in Ruination Theory[12] "The New Odysseus of 1966,"[13] prove this is because Adam Sandler and Hallmark are already ruined, and thus cannot ruin anything else[11]. They propose that anything that ruins Adam Sandler and Hallmark then, in a sense, "steals" or subsumes all of their ruinments.

$$\begin{tikzcd}
X \arrow[r, "\text{ruins}"] \arrow[d, dashed, "\text{ruins}"'] \arrow[rd, dashed, "\text{ruins}"] & \text{Hallmark} \arrow[d, dashed, "\text{ruins}"] \\
\text{Adam Sandler} \arrow[r, "\text{ruins}"] & \text{Movies}
\end{tikzcd}$$

They continue to generalize same principal to the general case of transitivity of ruinment. □

Exercise 2. *Prove your gender is trans-itive.*

Definition II.7. *Formal Ruinment is a preorder $\succ$ over $\mathfrak{R}$.*

Proof. Propositions II.2, II.3, II.4, II.5, II.6. □

III. Categorical[13] Ruination Theory

And, thus, we have formalized the ever-elusive notion of Ruinment in Ruination Theory. This, alone, would be enough to warrant an entire paper, maybe even 5, due to how novel this result is. But we, the searchers for Scientific Knowledge, would never—nay—*could never* do such a thing. And, thus, we continue ever onward into never-before-charted territory![14]

We continue with a brand-new angle on Ruination Theory:[15] Categorical[13] [16] Ruination Theory.

Theorem III.1. *Formal Ruinment over $\mathfrak{R}$ forms a category* [13] **Ruin**.

Proof. From Definition II.7 and the fact that all prosets are thin categories[12][13]. In this case, $\mathfrak{R}$ is the set of all objects[13] and each ruinment is a morphism[13]. □

And, now, we may bring in all sorts of results and generalizations found in Category Theory[13], such as the following corollaries:[17]

Corollary III.2. *Adam Conover is the initial[13] object* [13][18]*in* **Ruin**.

Proof. As **Ruin** is thin[13] (Theorem III.1), between any two objects[13] $A, B \in \mathfrak{R}$ there is at most one, unique morphism[13] between them, corresponding to the relation $A \succ B$. As Adam Conover ruins everything (The Taschetter–Conover Fundamental Lemma of Ruinment), there thus is a unique morphism[13] between Adam Conover and every object[13] in **Ruin** (Theorem III.1. Thus, Adam Conover is the initial[13] object[14][13] of the category [13]. □

IV. Waldstreicher–Equivalence

At this point, you may be thinking, "Wow! This paper has had so much succulent information for me to chew on with my Brain-teeth; how could there be any more?" To which I would respond, "What kind of person thinks such a thing? Like, seriously, 'Brain-teeth'? What is this, 1984?" To which you would probably reply "But SIGBOVIK didn't run in 1984!" and I would continue "The deadline is in one hour and I have to end this bit now." [19]

Definition IV.1 (Waldstreicher–Equivalence)**.** *A relation $\rightsquigarrow$ over a set S is Waldstreicher–Equivalent iff Adam Conover $\in$ S and $\forall s \in S : s \rightsquigarrow$ Adam Conover $\Leftrightarrow$ s is Adam Conover.*[20]

Theorem IV.2 (Waldstreicher–Equivalence Theorem)**.** *If somebody ruins Adam Conover, they are Adam Conover.*

Proof. Because Adam Conover is the initial[13] object[13] in the category[13] **Ruin** (Corollary III.2), there is a morphism[13] from him to every object. If somebody has ruined Adam Conover, then there must be a morphism[13] from them to him (Theorem III.1). Thus, this person is Adam Conover, up to unique isomorphism[13].[21] □

And, finally, the most important work of this paper, which motivated its very creation:

Corollary IV.3. *Maya Waldstreicher is Adam Conover.*

Proof. Maya Waldstreicher has ruined Adam Conover[18]. Thus, by the Waldstreicher–Equivalence Theorem, she is Adam Conover. [22] □

Exercise 3. *You just got Ruined! Tag your friends to totally Ruin them!*

[9] Ruination is preserved due to it producing *embodiments* of objects, which are naturally homo-by-definition.

[10] *Adam Sandler* ruins *movies* $\wedge$ *Hallmark* ruins *movies*

[11] pensive

[12] Though we use the term ruination in this paper, historical precedent dictates we call the field, in general, Ruination Theory.

[13] Copious research into the metamathematics of titling has provided us with the knowledge that 1966 is the birth year of Adam Sandler.

[14] Avast ye, for we are the bold and rousing crew of the Good Ship Ruinment!

[15] Man, writing this paper's abstract will be a cinch at this rate!

[16] In a Category[13] Theoretical sense, you cleanly Category[13] Theorists.

[17] Did we have a second one of these?

[18] We here at the Dissociation for Heresiographal Computation would like to sincerely apologize for objectifying Adam Conover. It was never our objective or desire to objectify a man. Though we have no intention of changing our behavior, we are deeply, deeply apologetic.

[19] We have been recently notified of the "final" deadline extension of submissions to SIGBOVIK 2022. Our stance on ending this bit where it stands has not been affected as a result of this sudden, deeply unexpected change.

[20] No, this isn't absurdly specific, what do you mean?

[21] TODO: find out what this means

[22] I can't believe I've met Adam Conover!

Appendix A
Not a Waste of Paper

Definition A.1 (Tascheter)**.** *A mathematical work is Tascheter iff it is Gödel-incomplete.*

Definition A.2 (Epilyssic)**.** *A mathematical work is Epilyssic (or Epic*[23]*) iff it is Gödel-complete.*

Definition A.3 (Evil)**.** *A situation is Evil iff it contains a mathematic work that is Tascheter and Epic.*

Lemma A.4 (Thesis)**.** *Appendix A of Jean–Yves "mad dog"*[15] *Girard's "Locus Solumn: From the rules of logic to the logic of rules" is incomplete.*

Proof. See Kurt Gödel's Incompleteness Theorems[16]. □

Lemma A.5 (Antithesis)**.** *Appendix A of Jean–Yves "mad dog"*[15] *Girard's "Locus Solumn: From the rules of logic to the logic of rules" is complete.*

Proof. Appendix A of Jean–Yves "mad dog"[15] Girard's "Locus Solumn: From the rules of logic to the logic of rules" is titled "A Complete Waste of Paper"[17].[24] □

Theorem A.6. *Appendix A of Jean–Yves "mad dog"*[15] *Girard's "Locus Solumn: From the rules of logic to the logic of rules" is Tascheter.*

Proof. Trivial from Lemma A.4. □

Theorem A.7. *Appendix A "As Seen On Lemmas A.4, A.5, and Theorem A.6" is Epic.*

Proof. Trivial from Lemma A.5. □

Theorem A.8 (The Thesis and Antithesis Come into Conflict)**.** *The current situation is Evil.*

Proof. Trivial from Theorems A.6 and A.7. □

Corollary A.9 (Synthesis)**.** *Jean–Yves "mad dog"*[15] *Girard and Kurt Gödel should have a fistfight.*

Proof. Trivial from Theorem A.8. □

Exercise 4. *Figure out why Jean–Yves "mad dog"*[15] *Girard is called "mad dog." Like, I'm genuinely curious.*

Appendix B
I Told You It Wasn't a Waste of Paper

This appendix, and future ones, have been brought to you by the "final" SIGBOVIK 2022 deadline extension.

We continue our study by looking at possible avenues for expanding the formal notion of Ruinment. In particular, there are two properties that are promising in the novelty they offer to Formal Ruinment: Strong Waldstreicher–Equivalence, and totality.

Definition B.1 (Strong Waldstreicher–Equivalence)**.** *A Waldstreicher–Equivalent relation $\rightsquigarrow$ on S is Strongly Waldstreicher–Equivalent iff $\forall A, B \in S$, $A \rightsquigarrow B \rightsquigarrow A$ implies A is B.*

Lemma B.2. *Formal Ruinment is Strongly Waldstreicher–Equivalent.*

Proof. Suppose we have two $A, B \in \mathfrak{R}$. If A ruins B and B ruins A, there must exist two morphisms[13]: one from A to B and another from B to A. Then, A is B up to unique isomorphism[13].[26] □

Lemma B.3 (Fistfight Lemma)**.** *Formal Ruinment is total.*

Proof. We generalize the dialectical process used in Corollary A.9. $\forall A, B \in \mathfrak{R}$ where $A \neq B$, there must be at least one contradiction between the two. Otherwise, all attributes of A and B are the same, and thus $A = B$ by extensional equality. The contradictions between A[27] and B[28] must eventually be resolved, according to Georg Wilhelm Friedrich Hegel's Dialectic Theorem[19].[29] As the time t approaches ∞, this will eventually result in a fistfight[21] [30] (Corollary A.9). The nature of any fight is such that there is a winner and a loser[22].[31] As the winner, by definition, ruins the loser, this means that either A ruins B, of B ruins A. Thus, Formal Ruinment is total. □

Even more important than the Fistfight Lemma, however, is the interpretations and perspectives it promises to bestow upon us. We demonstrate this below:

Corollary B.4. *Adam Conover could beat anybody in a fistfight.*

Proof. Trivial, by the Fistfight Lemma's interpretation of Ruinment-as-fistfights. □

Exercise 5. *Lose to Adam Conover in a fistfight.*[32]

And, so, we finally expand the definition:

Theorem B.5 (Tascheter's Totally Wicked Totality Theorem)**.** *Formal Ruinment is a total order.*

Proof. With the relation of "is" taken as equality[33], Proposition II.5 and Lemma B.2 imply that Formal Ruinment is antisymmetric. Any total (Fistfight Lemma) and antisymmetric preorder is a total order. □

Corollary B.6. *Every subset S of $\mathfrak{R}$ under Formal Ruinment has a minimum and maximum.*

[23] Not to be confused with an epimorphism[13], which is decidably not Epic, just like the people that named it that.

[24] One could object that the actual title of this appendix is "A Pure Waste of Paper". This has, of course,[25] already been refuted in a prior work [15].

[25] I am never wrong.

[26] TODO: complete TODOs. Y'know, usually I get syntax highlighting on my TODOs, but with my poor color scheme choice it usually just makes it invisible. Wait, is that why I can't see any of my TODOs?

[27] The thesis. Not an exponent.

[28] The antithesis. Also not an exponent.

[29] Probably; It's been more than a year since I read this. Who cares about proper citations anyway[20]?

[30] The synthesis. The notion of exponentiation is ill-defined in this context.

[31] The ∴|∴*er.*

[32] You $\xrightarrow{\text{lose}}$ Fistfight

[33] I mean, that's what is is.

Proof. Take some $s \in S$. Then, either s is a minimum or $\exists x \in S : x \succ s$.[34] Since $\mathfrak{R}$ is finite (Proposition II.3), we may count the number of elements that s ruins ($r_s \in \mathbb{N}$) and the number of elements it is ruined by ($d_s \in \mathbb{N}$). Notice, then, that $d_x < d_s$. We may induct on x to find that there must be some $m \in S$ such that $d_m = 0$, and thus that m is a minimum.

Similarily, take $t \in S$. Either t is a maximum or $\exists z \in S : t \succ z$. $r_t < r_z$. Induct on z to find that there must eventually be some n such that $r_n = |S|$, that is, n is the maximum. □

Appendix C
X–Treme Ruination

The Fistfight Lemma has enormous implications,[35] particularity in the methods used to deduce it. We thus provide an alternate view of Formal Ruinment, taking advantage of this perspective:

Definition C.1 (Obliteration)**.** *Obliteration is a total suborder of Formal Ruinment, where A ruins B iff A beats B in a fistfight.*

We continue by formalizing the intuitive notion of *Ruinment-as-fistfight* used in the previous appendix:

Corollary C.2 (Fistfight Isomorphism[36])**.** *Obliteration is equivalent to ruinment.*

Proof. By the Ruinment-as-fistfight interpretation of the Fistfight Lemma, if A ruins B, then A obliterates B. By definition of Obliteration, if A obliterates B, then A ruins B. □

While Obliteration[37] is just Formal Ruination, the ease of using this alternate perspective that it provides allows us to uncover a conspiracy hidden deep in each one of our hearts and minds.

Proposition C.3. *The following diagram[13] commutes[13] for Obliteration:*

$$\begin{tikzcd}
& \textit{Turtles} \arrow[d, leftrightarrow, "\textit{Refl}"] \\
\textit{Mario Mario} \arrow[ur, "\textit{Crushes}"] \arrow[r, "\textit{Obliterates}"] \arrow[dr, "\textit{Stomps}"'] & \textit{Turts} \arrow[d, leftrightarrow, "\textit{Refl}"] \\
& \textit{Turties}
\end{tikzcd}$$

Proof. We hold this truth to be self-evident[23]. □

While, horifically and conversely:

Proposition C.4. *The following diagram[13] commutes[13] for Obliteration:*

$$\begin{tikzcd}
& \textit{Turtles} \arrow[dl, "\textit{Crush}"'] \arrow[d, leftrightarrow, "\textit{Refl}"] \\
\textit{Wario Lastnameunknown} & \textit{Turts} \arrow[l, "\textit{Obliterate}"'] \arrow[d, leftrightarrow, "\textit{Refl}"] \\
& \textit{Turties} \arrow[ul, "\textit{Stomp}"]
\end{tikzcd}$$

Proof. Trivial, from prior[38] result[24]. □

Note that these two diagrams[13] are *the same* if the direction of the arrows[13] are flipped. Given the typical construction of the category[13] **Oblit** in light of the Fistfight Isomorphism, we could even say that, up to[13] the canonical contravariant[13] functor[13] **Oblit** → **Oblit**$^{\text{op}}$, Mario Mario *is* Wario Lastnameunknown. The contravariancy inherent in this statement's contingencies persuades us to re-inspect this notion. What we're really getting at is that Wario Lastnameunknown is Mario Mario's *opposite*. For lack of better terminology, we provide the following definition:

Definition C.5. *For every object*[13] A *in a category*[13] C*, we define the set of objects*[13] *wa-A in C^{op} by distinguishing the objects*[13] *in C^{op} equal to A up to the canonical contravariant*[13] *functor*[13] $C \to C^{op}$.

And, for consistency, we provide this complementary definition:

Definition C.6. *For every object*[13] A *in a category*[13] C *where wa-A exists, ma-$A = A$.*

Corollary C.7. *Wa- is involutive.*

Proof. Makes sense to me. □

Corollary C.8. *All elements of the set of wa-objects[13] are equal.*

Proof. Given A, B, C where $B, C \in$ wa-A. Then wa-B = wa-C = A. As wa-wa-B = B and wa-B = wa-C, wa-wa-$C = B$ so $C = B$. □

Thus, we treat any wa-A as a single object, rather than a set.

We continue to declare a few, key facts we may trivially determine from these formalizations:

- Waluigi is the opposite of Luigi.
- Mario[25] is Rio[26].
- A wawanut is just a nut.
- Mathematicians turn waffee into theorems.
- Wathematicians turn watheorems into ffee.
- The Philadelphia[39] chain Wawa does not exist.

Unfortunately, this avenue of study is useless and a dead end. In my 2 minutes of thinking of any use for this concept,

[34] By Tascheter's Totally Wicked Totality Theorem, if there is some y that isn't ruined by s, it must ruin s.

[35] In true categorical[13] fashion, the most-used results are lemmas.

[36] Also known as the *Tascheter–Hegel Correspondence*, *Fights-as-Ruins*, and *Ruinmental Twonitarianism*.

[37] Also referred to as *Crushing* or *Stomping*.

[38] Read: *unreleased.*

[39] Suck it, Yinzers.

I had absolutely zero ideas.[40] I wholeheartedly discourage anybody from looking into this further. In fact, please excise this appendix from your local copy of the proceedings and burn it.[41]

Notice 1. *This appendix has been retconned.*

Appendix C
Small Rigs: Over the Ruin Proving

At this point, you may be thinking "So, did they stop with the Category[13] Theory?[35] Haven't seen any for a while." Or, maybe, "Oh god, not this bit again. Please don't bring up Brain-teeth." Worry not, reader, for we are about to metaphorically "dive" back into the frey!

Corollary C.1 (Tascheter–Rumbletumble Corollary of Maximum Ruination)**.** *There is a final*[13] *object*[13] *in* **Ruin**.

Proof. By Corollary B.6, $\mathfrak{R}$ under Formal Ruinment has a maximum element. Thus, there is an object[13] F in **Ruin** that has a morphism[13] to every object[13] in the category[13]. F is, then, a final[13] object[13]. □

This corollary has fundamental results in Ruination Theory. There is something, somewhere, that *everything* has ruined.[36] Our good friend and reviewer Owen Rumbletumble postulates that this may be the Olympics[27], [37] however a proof of exactly *what* it is is outside the scope of this paper.[38]

Theorem C.2. *Finite products*[13] *and coproducts*[13] *exist in* **Ruin**.

Proof. $\forall A, B \in \mathfrak{R}$ construct the sets $P = \{p \in \mathfrak{R} : p \succ A \wedge p \succ B\}$ and $S = \{s \in \mathfrak{R} : A \succ s \wedge B \succ s\}$. We note that P has a unique maximum and S a unique minimum up to isomorphism[13] [39] due to Corollary B.6. These elements are, then, the product[13] and coproduct[13] of A and B by definition. □

Corollary C.3. $\mathfrak{R}$ *forms a lattice under products*[13] *and coproducts*[13].

Proof. Any poset under categorical[13] products[13] and coproducts[13] (Theorem C.2) forms a lattice[28]. □

We call the product[13] over this lattice (the meet) *Maceration*[40] and the coproduct[13] operation (the join) *Victimization*.[41] And, we shall now formalize these definitions:

Definition C.4 (Maceration)**.** *Maceration is the product[13] in* **Ruin**.

Definition C.5 (Victimization)**.** *Victimization is the coproduct[13] in* **Ruin**.

For intuition's sake,[42] we give the following lexical shorthands: the macerator of $A, B \in \mathfrak{R}$ (where A and B are the macerates) is something that ruins both A and B, but is ruined by all other things that also ruin A and B. The victim of A and B (where A and B are the victimators) is something that both A and B ruins, which also ruins everything else that A and B also ruins.

Lemma C.6. *Maceration distributes over victimization.*

Proof. Denote maceration as $\times$ and victimization as $+$. Then, for distributivity to hold, $\forall A, B \in \mathfrak{R} : A \times (B + C) = (A \times B) + (A \times C)$. We know that either $A \times (B + C) \succ (A \times B) + (A \times C)$, $(A \times B) + (A \times C) \succ A \times (B + C)$, or both when equal (Tascheter's Totally Wicked Totality Theorem). We define $L = A \times (B + C)$ and $R = (A \times B) + (A \times C)$, then split based on whether the first or second statement is true:

$L \succ R$**:** Then R must be the victim of L and itself:

$$\begin{tikzcd}
A \times B \arrow[r] \arrow[dr] & (A \times B) + (A \times C) \arrow[d] & A \times C \arrow[l] \arrow[dl] \\
 & X & \\
A \times (B + C) \arrow[r] \arrow[ur, dashed] & (A \times B) + (A \times C) \arrow[u] \arrow[loop right] &
\end{tikzcd}$$

By the Ruinment-as-fistfights interpretation of the Fistfight Lemma, this implies that R intentionally lost, thus stripping L of a true victory.[43] Therefore, L didn't truly win, and so $R \succ L$. By the Strong Waldstricher–Equivalence of Formal Ruinment, $L = A \times (B + C) = (A \times B) + (A \times C) = R$.

$R \succ L$**:** Then, by transitivity of Ruinment (Proposition II.6), $R \succ A$ and $R \succ (B + C)$. As L is the macerator of A and $B + C$,[44] this results in a contest for the throne of L. R attempts to take over the position of macerator, [45] eventually and inevitably failing due to the entrenched power of L.[46] Thus, in a second coming [47] of Wilhelm Georg "Heg-Man" "Dr. H" Hegel[29], $L \succ R$, and so by the Strong Waldstreicher–Equivalence of Formal Ruinment, $L = A \times (B + C) = (A \times B) + (A \times C) = R$. □

Theorem C.7. *Maceration and victimization form a rig over* $\mathfrak{R}$*, with respective identities of the Olympics (Used as placeholder due to the Rumbletumble Postulate) and Adam Conover.*

Proof. First, note that the join and meet of a lattice (Corollary C.3) are associative and commutative, that the initial[13]

[40] Don't think I could fit it into my abstract either.

[41] You $\xrightarrow{\text{obliterate}}$ SIGBOVIK 2022 Conference Proceedings

[35] 'Did they just cite my own words?'

[36] I will withold the jokes, as this is extremely serious.

[37] I was lying.

[38] Read: a book.

[39] Well, I know what *this* means at least.

[40] We also suggest the alternate names of *Macerment*, *Mutilation*, *Mutilment*, and *TwoBirdsOneStoning*.

[41] Or *Double Homicide*.

[42]

[43] Egads!

[44] Thesis.

[45] Antithesis.

[46] Synthesis.

[47] Zounds!

object[13] is the identity of the coproduct[13], and that the final[13] object[13] is the identity of the product[13] (Theorem C.2. Then, victimization and maceration must form monoids. Further, victimization is a Mathematical[48] monoid. As maceration further distributes over victimization (Lemma C.6), they form a rig. □

Theorem C.8. Ruin *is a distributive, and thus rig, category*[13].

Proof. Trivially, by Theorem C.2, Corollary C.3, and Lemma C.6. □

Appendix D
Perhaps the Most Unusual Result Ever to Come from Ruination Theory

And, so, we finally reach what is perhaps the most unusual result ever to come from Ruination Theory:

A. What is Perhaps the Most Unusual Result Ever to Come from Ruination Theory

Declaration 1 (Perhaps the Most Unusual Result Ever to Come from Ruination Theory)**.** *We're all complex.*[49] *:)*

Proof. We may treat all objects[13] in a rig category[13] as complex numbers as long as certain[50] conditions are met[31]. As every person is an object[13][51] in **Ruin**, every person is complex. □

Almost as complex as an orange.

Exercise 6. *Eat an orange. You deserve it.*

REMEMBER TO WRITE ABSTRACT!

References

[1] Redacted in order to conform to HIPPA. Summer 2020.

[2] Wikipedia, "Library of Alexandria." 1-Mar-2022. From https://en.wikipedia.org/wiki/Library_of_Alexandria

[3] truTV, "Adam Ruins Everything." 29-Sep-2015.

[4] Alledgedly. 300,000 years ago.

[5] Steve Waddington, CIO, CEO, Gemologist, Proud Father[52] of Three, Answer to "Is there a physical limit on information density?" on Quora. "5 years ago." From https://quora.com/Is-there-a-physical-limit-on-information-density

[6] I don't need to cite Arabic, that's just how it works. 1-Apr-2022.

[7] Know Your Meme, "You Either Die A Hero, Or You Live Long Enough To See Yourself Become The Villain." Jun-2019. URL not included because I'm too lazy to format it.

[8] Sexypedia Wiki. "The Starry Night." 8-Jul-2021. From https://web.archive.org/web/20210708181835/https://tumblrsexymen.fandom.com/wiki/The_Starry_Night

[9] My roommate, "You can turn anything into an idealized, sentient Tumblr Sexyman." A few seconds ago.

[10] Good Movies List, "Good Movies List — Best movies of all time to watch on Netflix and Amazon." 2020. From https://goodmovieslist.com

[11] There's only, like, one copy left after that whole library fire? And I think I lost it in my pile of spam mail, bills, and old stolen library reference cards. Uhh, 2000-something? Somewhere in the 21st century.

[12] nLab, "partial order." 25-Mar-2022. From https://ncatlab.org/nlab/show/partial+order

[13] Wikipedia, "Abstract nonsense." 3-Nov-2021. From https://en.wikipedia.org/wiki/Abstract_nonsense

[14] nLab, "initial object." 25-Mar-2022. From https://ncatlab.org/nlab/show/initial+object

[15] Reginald J. Qnuth, "A Systematic evaluation of the Observed Degradation of Typesetting Technology in the 20th Century." 3-Mar-2007. From SIGBOVIK 2007.

[16] I don't speak German. Maybe try Exercise 1? Future.

[17] Jean–Yves "mad dog"[15] Girard, "Locus Solumn: From the rules of logic to the logic of rules." 5-Jul-2001. From *Mathematical Structures in Computer Science.*

[18] Maya Waldstreicher, "Adam Ruins Everything is bad." 26-Mar-2020.

[19] George Ciccariello-Maher, "Decolonizing Dialectics." 2017. From Duke University Press.

[20] sha256sum, "f299 a401 e4ee ffac df5d b8d1 090c c4e9 aecf 30aa 3107 c828 54d8 b255 c2be ae75". 207̃012.

[21] Phindoll, "Well, it's a fistfight, so prepare to scrap! Ker-pow!" 15-Feb-2021. From "∃NA: Temptation Stairway."

[22] Big Rigs: Over the Road Racing, "YOU'RE WINNER !" 20-Nov-2003.

[23] Phil Jamesson, "Mario, the Idea vs. Mario, the Man." 18-Feb-2022. From Philosophy 101: Midterm 1.

[24] Nintendo Entertainment Analysis & Development Division Group No. 1, "Mario Kart DS." 14-Nov-2005. From GameStop Corp.

[25] Somehow Licensed by Nintendo Co., Ltd., "Super Mario Bros." 28-May-1993.

[26] The Creators of Ice Age, "Rio." 15-Apr-2011.

[27] Owen Rumbletumble. "The Olympics used to be about naked people doing cool sports for fun. Cut to today, it sucks, everyone takes it way too seriously and absolutely nobody is naked." 26-Mar-2022.

[28] nLab, "lattice." 26-Mar-2022. From https://ncatlab.org/nlab/show/lattice

[29] Luna A. تـــاشـــتر , "On Ruinment: Ruination Theory and its Consequents"[30]. 7-Apr-2022. From SIGBOVIK 2022.[53]

[30] Luna A. تاشتر, "On General Recursion in SIGBOVIK papers"[29]. Preprint. From under my bed.

[31] Marcelo Fiore, Tom Leinster, "Objects of Categories as Complex Numbers." 30-Dec-2002. From https://arxiv.org/abs/math/0212377

[48] As commutative groups are named Abelian after Abel Abel (brother of Cain Abel), we call commutative monoids Mathematical after Eliza L. N. B. Mathematics, founder of mathematics[29].

[49] [laughtrack]

[50] Irrelevant.

[51] Our behavior remains unchanged.

[52] "Fatber" in some circles, due to typo in draft of this paper.

[53] Perchance.

Astrophysics

18 Method and Tool for Estimating the Mass of the Black Hole Located in the Office of Immigration, Refugees and Citizenship Canada Causing a Supermassive Time Dilation in the Visa Extension Process

Étienne Trottoir-Barré, Richard von Pamplemousse and Jessica G. Lasso

Keywords: black hole, abracadabra, hyper driven devices, natural language processing, Chevy Tahoes

19 Black Hole Computation

Matias Scharager

Keywords: monad, computation, compilation, computational trinitarianism, parallel universe, parallel computing

20 Schrödinger's SAT: Generalizing Quantum Bogosort to Prove P = NP Under Many-Worlds Quantum Mechanics

Melody Horn

Keywords: satisfiability, time complexity, revolutions in our understanding of computing, unsolved problems in millennium prize eligibility

21 Solving Double Execution of Java's paint() Method by Counting Down to the Heat Death of the Universe (plus language compendium)

Braden Oh, Vedaant Kuchhal, Junseok Kang, Andrew Mascillaro, Justin Kunimune, Shashank Swaminathan, Devlin Ih, Elias Gabriel, Audrey Lee, Colin Snow, Hyunkyung Rho, Ben Morris, Solomon Greenberg, Mahima Beltur, Anusha Datar, Jonathan Kelley, Pranavi Boyalakuntla, Xander Hughes, Devyn Oh, Aidan Schmitigal

Keywords: Efficient, Effective, Compendium, Languages, Heat death, Overkill

Method and Tool for Estimating the Mass of the Black Hole Located in the Office of Immigration, Refugees and Citizenship Canada Causing a Supermassive Time Dilation in the Visa Extension Process

Étienne Trottoir-Barré[1], Richard von Pamplemousse[2], and Jessica G. Lasso[3,4]

[1]The Streets of Montréal, Ó Cánâdà
[2]Enshrouded within the Twisted Linens of Madness, Montréal
[3]Vermont College of Parks and Recreation, Montpelier, VT, Greater South East Canada
[4]Ted Lasso School of Business and Coaching, Montpellier, France, Far East Canada

Abstract

Despite advances in business process modeling, process discovery and just-better-than-nothing SLAs, obtaining and extending a working visa as a foreign academist in Canada is borderline impossible. This problem has resisted significant technological innovations, such as *the Internet* and *crypto-coins*, and nothing seems to speed up the visa process. We conjecture that the dense core of this problem is a *black hole* located in the office of Immigration, Refugees and Citizenship Canada (IRCC). This black hole warps the very fabric of the space-time continuum within the office such that the time dilation results in observable differences. Here we crudely estimate the mass and radius of this black hole, we call *IRCC-*∞ based on maths, science, and carefully sampled data from ~~two~~ ~~three~~ ~~two~~ ~~people~~ one person. We introduce an open-source modeling and simulation tool as well as our signature Gradient Condescend$^{\text{TM}}$ algorithm that shows excellent scalability in space and time, unless carried out in the critical vicinity of a black hole. In an effort to meet the scientific standards of the 2020's, we also point out that *the COVID-19 pandemic has seriously exacerbated this issue*, as evidenced by the IRCC website: ”Due to the impacts of COVID-19, we (i) can't process applications normally, (ii) give accurate processing times for most applications”.

Keywords include but are not restricted to: black hole, abracadabra, hyper driven devices, natural language processing, Chevy Tahoes

1 Introduction

Black holes [Pedia, 2022b] have been identified by many empirical [Trottoir-Barré et al., 2016] and theoretical [von Pamplemousse and Ansymov, 2019] studies, esoteric ponderings [Nuseibeh and Easterbrook, 2000], and transformationally innovative patents [San Fearlow and Ansymov, 2015] as the main challenge in delivering upon previously agreed service-level- agreements (SLAs). Black holes have the ability to significantly dilate time. Unfortunately the known lemma of Newtonian physics ”time is money” does not apply in relativity.

A pertinent example of time warping related issues is the visa extension process as a foreign academist in Canada. Our empirical data set sampled from one person and confirmed by anecdotal evidence shows significantly exceeded tentative processing times. While such discrepancies are often explained by inefficient business processes, additional evidence suggests that the most plausible explanation in this case is a rogue black hole in the middle of the office of the Immigration, Refugees and Citizenship Canada (IRCC), we call *IRCC-*∞. The evidence includes, but is not restricted to the following statement by the IRCC website.

”*We can't process applications normally and can't give accurate processing times for most types of applications. If you contact us, our Client Support Centre agents do not have additional information on processing times. We can't give you more information than what is already available in your account.*”[*]

[*] https://www.canada.ca/en/immigration-refugees-citizenship/services/application/check-status.html. To reproduce the experiment, select ”Work permit”, ”Work permit inside Canada (initial, extension or change of conditions)”, ”Online”.

Clearly, the fabric of the space-time continuum is so warped in and around IRCC offices that nothing, not even *information* can exit ("can't give accurate processing times for most types of applications"). Moreover, there is a possibility that IRCC agents are situated *within* the said black hole, suggested by their lack of ability to *receive* information ("our Client Support Centre agents do not have additional information on processing times"). The whole situation is supremely exacerbated by the ambient Schrödinger phenomenon [Schrödinger, 1935]: "**If you contact us**, our Client Support Centre agents **do not have additional information** on processing times." We are, unfortunately, unaware whether the agents possess this information **if** we do **not** contact them. It seems to be likely that the information is destroyed upon observation.

To cope with this wicked problem [Rittel and Webber, 1974], we reinvented multiple facets of science from theoretical, applied and fictional physics to industry-scale inept software buffoonery and the semantics of reification in Žižek's neo-marxism [Zizek and Daly, 2004].

Contributions. The contributions of this paper include, but are not restricted to the following.

- We formalize black holes as an algebraic structure.
- Based on this structure, we propose a methodology to calculate the mass of the *IRCC*-∞ black hole, called the Gradient Condescend^{TM}.
- We present a tooling to operationalize the method, and en route to developing it, we simply redefine software engineering by inventing software development by Natural Language Processing and low-code platforms, effectively rendering the past five decades of software engineering travesty obsolete.
- We suggest a solution to the problem at hand that does not work, by calculating the number of Chevy Tahoes required to tow the black hole out of the office.

2 Related Work

To situate our work amongst the literature, we have conducted a rather elaborate search on Wikipedia to figure out what a black hole is [Pedia, 2022b]. The results were quite interesting but they are omitted here for no reason.

Instead, we wish to question the repeated insistence of reviewers that we provide related work at all. First, related work is kind of boring to read (unless it cites us). Instead, go play *Papers, Please*[†], which is a fun game related to immigration.

Second, we state that all of our ideas are simply incomparable. The very words themselves are like piercing needles of insight which slice through the eyeballs and forcibly inject reason upon the mind. How could the slurred ramblings of others, whose ideas writhe upon the dirt like diseased worms, approach the glorious word pillars of truth and beauty which we erect before you?

Third, all scientific works such as this one are majestic poems in their own right, and should not be trivialised with comparisons. We ask whether one compares each endless wave crashing upon a child's sandcastle, or the mournful trills of each songbird secluded deep within the woods, or late-night hot-dogs from dodgy street vendors when you've had a few too many drinks to forget another paper rejection. Are not each of these notes in the great background soundtrack of life, just as each paper randomly found on Google Scholar is a weave in the great scientific tapestry? Are not all scientists kin on this hurtling fragile orb through space? As such, we reject the false tribal divisions cast upon us by the putrid reviewer #2, who aims only to incite hatred and envy of those who have actually performed proper experiments and ethical review. Nay, we say! We do not cower in the face of your pathetic meaningless cries for related work. All publications are related, just as we are all related in our quest for scientific truth which contains enough buzzwords to be funded.

3 Background

In this section, we give a brief overview of the key concepts of our work.

[†] https://store.steampowered.com/app/239030/Papers_Please/

3.1 Black Holes

A *black hole* is formally defined as a 4-tuple $\mathfrak{B} = (x, y, z, t)$. That is, the black hole $\mathfrak{B}$ is always (t) somewhere (x, y, z). According to Hawking [Hawking, 2015], for an asymptotically flat spacetime, a supertranslation α shifts the retarded time u to $u' = u + \alpha$, where α is a function of the coordinates on the 2-sphere. A 2-sphere is differentiable everywhere, and their derivatives can be calculated by using the 2 Chainz rule [Pedia, 2022c], developed independently by Leibniz, l'Hôpital, but mostly by Tauheed K. Epps [Pedia, 2022a]. (Discrete 2-sphere derivatives are safe to be solved by the Ja Rule [Pedia, 2022e].) We make use of the differentiability of the 2-sphere in Section 4.1, where we employ a Gradient Condescend$^{\text{TM}}$ method to find the mass of the *IRCC*-∞.

3.2 Time Dilation

Time dilation is the difference in the elapsed time as measured by two clocks [Einstein and Davis, 2013, Pedia, 2022f], assuming you can afford two clocks in this economy.

Lemma 3.1 *The time dilation measured between two clocks is equivalent to the time dilation measured by one clock first measuring the non-dilated time, and then measuring the dilated time.*

The proof is left as an exercise for the reader. We suggest the reader to begin their proof by having their twin be an astronaut [Garrett-Bakelman et al., 2019].

3.3 Hyper-driven Devices

Hyper-driven devices were first introduced by Bovik [Bovik, 1975]. Hyper driven devices are equipped with a propulsion system that enables travel in the hyperspace. As explained in Section 3.1, our approach leverages the fact that a black hole $\mathfrak{B} = (x, y, z, t)$ is situated in a 2-sphere.

Lemma 3.2 *A hyperspace is generalized n-sphere.*

Corollary 3.2.1 *To effectively calculate the derivative of the hyperspace of a hyper driven device, we need to apply the* $\frac{N}{2}$ *Chainz rule.*

3.4 Canada

Canada is an awesome country, yet slightly chilly at times. Figure 1 shows the majestic Canadian flag. Historical Canadian records [McKenzie and McKenzie, 1980] state that those of sufficient Canadianness can scratch-and-sniff this figure and detect the faint whiff of maple syrup. Future work will determine the applications of this Canadian detection method to speedup the visa application time.

Figure 1: Behold the great Canadian flag. Humble yet glorious. So simple yet impossible for a child to draw. When Canadians rule the world, you'll all be sorry.

Lemma 3.3 *The maple leaf in the Canadian flag is* <u>*red*</u>*.*

Corollary 3.3.1 *The maple leaf in the Canadian flag cannot be a* <u>*black*</u> *hole.*

4 Estimating the Mass and Radius of *IRCC*-∞

In this section, we define a scalable method for estimating the mass and radius of *IRCC*-∞. The goal is to estimate the mass of *IRCC*-∞ based on the originally set *tentative execution time* of the visa processes, and the *observed eventual* time it took to grant the said visas. For methodical soundness, this shooting problem is solved while consuming shots.

4.1 Algorithm: Gradient Condescend$^{\text{TM}}$

According to W. Pedia et al. [Pedia, 2022d], in mathematics, gradient descent (also often called steepest descent) is *a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.* We establish, that we are firmly in mathematics and therefore, this definition applies. Our modified version of the algorithm, the *Gradient Condescend^{TM}* shows superior scalability and convergence by introducing a tolerance range ϵ for a candidate solution. Every candidate solution c is deemed to be acceptable if it fits the $(c - \epsilon, c + \epsilon]$ range, where $-\infty \leq \epsilon \leq \infty$. This definition implies that every candidate solution works as an appropriate approximation of the solution. Therefore, as the algorithm progresses and we accumulate more and more spectacular solutions, we get more confident about our intellectual superiority and consequently, more condescending to anyone having issues solving the problem at hand.

Lemma 4.1 *For $\epsilon = -\infty$ the Gradient Condescend^{TM} method does not converge.*

The proof is left as an exercise for the reader. Please, do not attempt this exercise at home without the supervision of a professional trainer.

The Gradient Condescend$^{\text{TM}}$ requires the underlying structure being differentiable everywhere. If the underlying structure is not differentiable somewhere, the underlying structure is transformed in a way that it becomes differentiable everywhere.

The steps to execute the Gradient Condescend$^{\text{TM}}$ are the following.

Algorithm 1 Gradient Condescend$^{\text{TM}}$

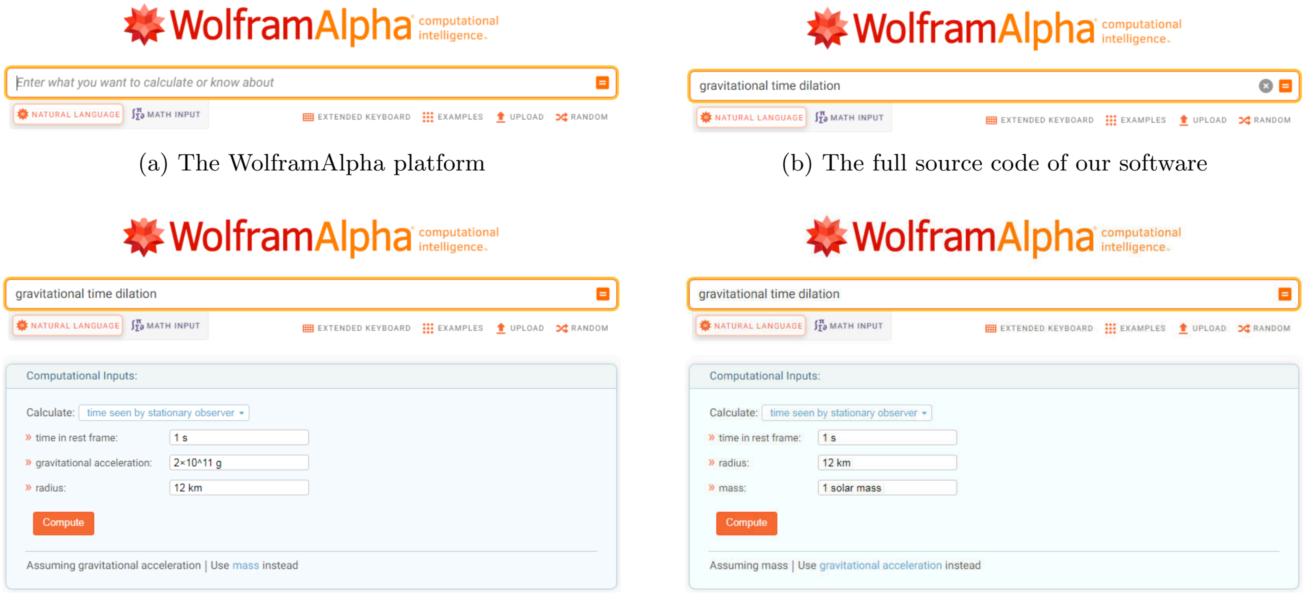
```
Require: $n \geq 0$
Ensure: $t_0, r \in \mathbb{R}$
Ensure: $t' = 0$
Ensure: $m = 1\ M_\oplus$                                   ▷ $M_\oplus$: Earth mass
  while $|t_0 - t'| > \epsilon$ do                 ▷ $t_0$: target time, $t'$: current solution
    t' ← simulate(r, m)
    if $t' > t_0$ then
      m ← m++
    else
      m ← m−−
    end if
  end while
```

4.2 Simulation Tool Development by Natural Language Processing

To model and simulate the problem, and to enact the Gradient Condescend$^{\text{TM}}$ algorithm, we developed a suitable software tool. Abandoning traditional software engineering methodologies and rapidly accelerating towards a glorious new future, we developed the tool purely by Natural Language Processing, as shown in Figure 2.

We build our tool on the popular WolframAlpha low-code [Sahay et al., 2020] platform (Figure 2a). Low-code platforms, as the name suggests, allow lowly qualified commoners/peasants – i.e., anyone who does not consider C++ and vim [Pedia, 2022g] the superior programming language and environment for *any* software engineering task from deep space telescope control to scripting Excel sheets – to somehow magically create software. This is potentially undesirable, as software programmers must be made to suffer through learning about pointers and assembly programming. However, low-code is a really big buzzword right now, so we'll do it for the keyword.

The full source code of the tool shown in Listing 1 is entered into the platform (Figure 2b). Subsequently, the *Enter* is pressed, and the platform generates the tool (Figure 2c). Of course, this is a rudimentary end-product, further evidencing how far AI currently is from world domination. Our superior human intelligence is required to improve the tool to be able to work with masses (Figure 2d).



Enter what you want to calculate or know about

(a) The WolframAlpha platform

gravitational time dilation

(b) The full source code of our software

gravitational time dilation

Computational Inputs:

Calculate: time seen by stationary observer

» time in rest frame: 1 s

» gravitational acceleration: 2×10^11 g

» radius: 12 km

Assuming gravitational acceleration | Use mass instead

(c) Generated tool with an absolutely preposterous default configuration. Of course I want to use *mass* instead

gravitational time dilation

Computational Inputs:

Calculate: time seen by stationary observer

» time in rest frame: 1 s

» radius: 12 km

» mass: 1 solar mass

Assuming mass | Use gravitational acceleration instead

(d) Properly configured tool thanks to the superior human-intelligence-in-the-loop

Figure 2: End-to-end development process of the tool

Listing 1: Full source code of the tool.

```
gravitational time dilation
```

The tool is available as an open-source project.[‡]

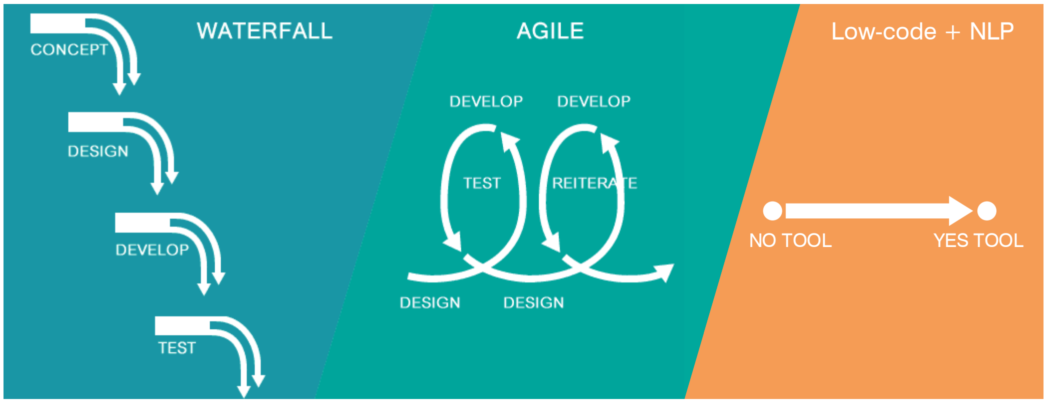


Figure 3: The hassle of traditional and agile software engineering vs *low-code + NLP*

Figure 3 compares all the hassle of traditional software engineering (such as waterfall) and agile approaches, with the beauty, elegance, and simplicity of our approach. In Section 6 we will consider rewriting the whole paper focusing on this contribution alone. Also, in future work we will offer 'YES TOOL' seminars to convince middle-managers that we should be hired as consultants. Many profits await.

As a caveat, we note that we wanted to work with a tool that calculates dilation in days, but only found this one that converts everything to minutes. Being the experienced researchers we are, we immediately recognized the illusion of requirements [Ralph, 2013] and settled for what we had.

[‡] https://www.wolframalpha.com/input?i=gravitational+time+dilation&assumption=%7B%22F%22%2C+%22TimeDilationGravitational%22%2C+%22r%22%7D+-%3E%2212+km%22&assumption=%7B%22FS%22%7D+-%3E+%7B%7B%22TimeDilationGravitational%22%2C+%22to%22%7D%7D&assumption=%7B%22F%22%2C+%22TimeDilationGravitational%22%2C+%22to%22%7D+-%3E%221+s%22&assumption=%7B%22F%22%2C+%22TimeDilationGravitational%22%2C+%22M%22%7D+-%3E%221+solar+mass%22

4.3 Calculation and results

For our calculations, we fix the following variables:

- t = 190 days,
- r = 0.535 m

According to the Canadian Centre for Occupational Health and Safety (CCOHS), “a basic workstation - such as a call center” has “107-132 cm x 152-183 cm” ”””Minimum”” ””””Requirement”””” Ranges” [CCOHS, 2022]. Other office types have larger requirement ranges, thus, we choose the basic Canadian workstation as the basis of our calculations to minimize existential threats to validity. Given a 107cm×152cm office, the radius of *IRCC*-∞ cannot be larger than 53.5cm, or, 0.535 meters.

In accordance with Algorithm 1, we will use the fixed t and r values to calculate candidate solutions using the tool. We continue the calculations until we find a configuration that results in 190 observed days outside of the IRCC office, and 119 observed days inside the IRRC office. (For the detailed data, see the replication package in the Appendix, and Table 2.) Table 1 shows the steps of calculation. Columns m and t' show the estimated mass of *IRCC*-∞, and the resulting observed time in the rest frame, respectively. The next column shows whether t' as a solution is acceptable or not, disrespectively.[§] The acceptability of the solution is determined by checking whether $t' \in [t' - \infty, t' + \infty]$, as discussed in Section 4.1.

Table 1: Steps of calculation

m	t'	Acceptable?	required action
1 $M_\oplus$	(radius within the Schwarzschild radius)	Kinda (See Lemma 4.1.)	▲increase
10 $M_\oplus$	173 days 12 hours 50 minutes	First Yes	▲increase
40 $M_\oplus$	110 days 6 hours 26 minutes	Multi Yes	▼decrease
35 $M_\oplus$	123 days 2 hours 12 minutes	Mo-mo-mo-monster Yes	▲increase
37.5 $M_\oplus$	116 days 20 hours 30 minutes	UnstoppabYes	▼decrease
36.5 $M_\oplus$	119 days 9 hours 21 minutes	Ultra Yes	▲increase
36.75 $M_\oplus$	118 days 18 hours 16 minutes	Wicked Sick Yes	▼decrease
36.7 $M_\oplus$	119 days	Holy Yes!	🏆 Draft Turing Award acceptance speech

The mass of black hole *IRCC*-∞ is estimated to be 36.7 Earth masses.

5 Discussion

In this section, we discuss the potential mitigation and solution strategies to the problem outlined previously.

5.1 Towing *IRCC*-∞ with a fleet of 2021 Post-COVID Chevy Tahoes

The first solution we investigate is towing *IRCC*-∞ out of IRCC's office. We choose a 2021 Post-COVID Chevy Tahoe as the basis of our assessment. Such a vehicle, equipped with a 5.3L V8 engine, possesses a 8400lbs towing capacity [Sansing, 2022], equivalent to 7.5902275769746E-26 Earth Masses ($M_\oplus$). To mitigate the threats to external validity, we also calculate that 8400lbs are equal to 2.3873263157895E-28 Jupiter Masses. This way, potential IRCC offices which are placed on the surface of the Jupiter can be handled as well.

To completely bypass *threats* of external validity (as this sounds scary), and turn them into *opportunities* of external validity, we calculate that 8400lbs are equal to 2.7315930097516E+26 Atomic Mass Units. Since everything is made of atoms, this formula now makes our approach applicable to everything, including but not restricted to: planets, Universal and Disney Turing Machines, left-handed optical mice, vilified programs [Bovik and Rude, 1986], and hyper-driven devices [Bovik and Fakir, 1988].

According to the well-established domain-specific safety best practices [Blue, 2021], one should never exceed the vehicle's towing capacity, and it's best to never come within 10% of that total. Thus, we calculate, that the towing capacity τ is equal to 8400×0.9 lbs = 6.83120481927714E-26 $M_\oplus$.

[§] As demanded by the Gradient Condescend$^{\text{TM}}$ algorithm.

We use the following formula to calculate the number of 2021 Post-COVID Chevy Tahoes required to tow *IRCC-∞*:

$$\text{🚘} = \frac{m_{IRCC\text{-}\infty}}{\tau} \times \frac{1}{0.9}, \qquad (1)$$

where

- $m_{IRCC\text{-}\infty} = 36.7\ M_{\oplus}$ (Section 4.3),
- $\tau = 6.83120481927714\text{E-}26\ M_{\oplus}$,
- and $\frac{1}{0.9}$ is the safety measure.

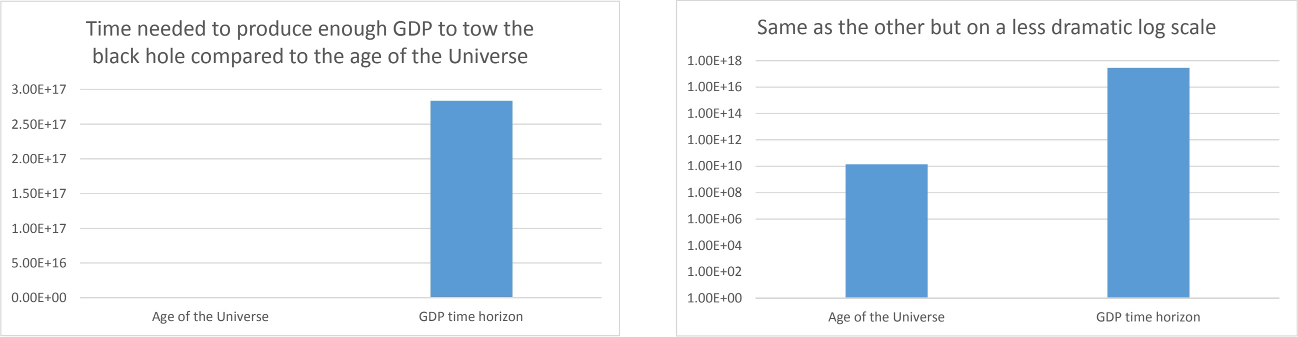
We need 🚘= 5.3724051570574188685840817467 02E26 2021 Post-COVID Chevy Tahoes to tow black hole *IRCC-∞*.

Cost assessment

We estimate the price of a new 2021 Post-COVID Chevy Tahoe to be USD50.000[¶]. This gives us USD2.7E31 as the cost of towing the black hole. Material costs of connecting the cars, inflation and static friction are omitted.

Time assessment

The global GDP in 2021 amounted to USD95E12. A conservative estimation gives us 2.84E17 years of global production to pay for the Chevy Tahoes required to tow the black hole. For comparison, the Universe is estimated to be 13.77E9 years old. For visual comparison, see Figure 4. For causing panic (especially for marketing reasons), please, refer to Figure 4a. For confusing people with scales no one can relate to (especially for scientific reasons), please, refer to Figure 4b.



(a) Comparison of the time needed to produce enough GDP to tow the black hole, and the age of the Universe (dramatic scale)

(b) Comparison of the time needed to produce enough GDP to tow the black hole, and the age of the Universe (logarithmic scale)

Figure 4: Comparison of the time needed to produce enough GDP to tow the black hole, and the age of the Universe

Feasibility assessment

It does not seem to be feasible to solve the problem at hand by towing black hole *IRCC-∞* out of the IRCC office in the foreseeable future. Or in the non-foreseeable, for that matter.

[¶] Equal to about 62,700 CAD or quite a bit of Canadian Tire Money (CTM). Readers should however note that CTM can be deceivingly valuable [Hrvatin, 2017]

5.2 Constructing another black hole that will swallow *IRCC*-∞

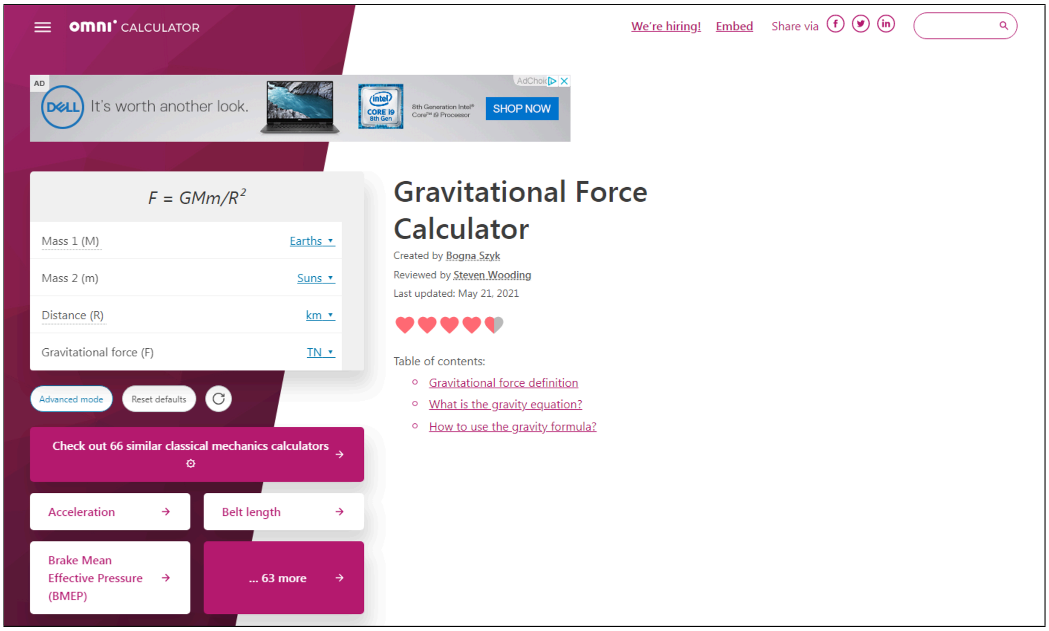
After showing that towing the black hole with a fleet of Chevy Tahoes is not feasible, we could investigate a more radical and innovative option: constructing a black hole that can attract *IRCC*-∞, pulling it out of the office, and eventually consuming it. Such an approach opens up great opportunities in strategically placing the newly created black hole to warp or stop time where it is actually useful, for example, in the tax office.

This investigation requires thorough consideration of the location of the new black hole, thus, we only outline the solution here and will elaborate on it in future work.

We suggest using the domain-specific tool from the award-winning Omnicalculator suite[‖], that has been recognized as the ***Nice Gravitational Force Calculation Tool*** of 2020 by Jeff Mangum [CHIRAG, 2020].

Figure 5 shows the uniquely supreme integrated user experience provided by the graphical user interface of the Omnicalculator suite. While calculating, the suite also allows the user to carry out the following tasks.

- Learn about the newest developments of the hardware industry and the specifications of an unspecified DELL computer, probably a laptop.
- Apply for jobs. (*We're hiring!*)
- Check out 66 other calculators of the suite.
- Learn about the 4.5 ♥ rating of the calculator, suggesting a higher-than-average computational precision and faster-than-IRCC execution.


omni° CALCULATOR
We're hiring!
Embed
Share via
AD
DELL
It's worth another look.
intel CORE i9 8th Gen
8th Generation Intel® Core™ i9 Processor
SHOP NOW
AdChoices
Advanced mode
Reset defaults
Check out 66 similar classical mechanics calculators
Acceleration
Belt length
Brake Mean Effective Pressure (BMEP)
... 63 more
Table of contents:
Gravitational force definition
What is the gravity equation?
How to use the gravity formula?

$F = GMm/R^2$

Mass 1 (M) Earths

Mass 2 (m) Suns

Distance (R) km

Gravitational force (F) TN

Gravitational Force Calculator

Created by Bogna Szyk

Reviewed by Steven Wooding

Last updated: May 21, 2021

Figure 5: Integrated user experience in the Omnicalculator suite, the first known Massively Multitasking Online Result-Producing Gadget (MMORPG)

This unique combination of features renders the Omnicalculator suite a Massively Multitasking Online Result-Producing Gadget (MMORPG), the first of its kind. Consequently, calculating the gravitational force required to pull *IRCC*-∞ out of the office becomes a ridiculously trivial task.

6 Scope of the Paper

After considering rewriting this paper to focus more on the groundbreaking results in Figure 3, we decided to keep the scope of the paper as it is and rewrite it after the conference.

[‖] https://www.omnicalculator.com/physics/gravitational-force

7 Conclusion

The word *conclude* comes from two Latin components: a) *con* meaning *completely*, and b) *claudere* meaning *to enclose*. How fitting. Just as you, dear reader, are ensnared amongst the written traps of this overly verbose conclusion, and just as we are engulfed in the maelstrom of the publish-and-or-perish Faustian bargain, so too are the poor office workers at Immigration, Refugees, and Citizenship Canada (potentially) trapped beyond the event horizon of a black hole.

In this paper we formalized black holes, introduced methodologies, tools, and opened up novel ways of thinking about computer science at large. Our results show that IRCC is beyond repair.

Our future studies will involve salami-style publishing of many conceptual reference frameworks regarding this important issue. Drinks are also foreseen.

8 Post-submission conclusion

In an unexpected turn of events, the visa extension investigated in this paper got approved two days after the submission.

APPENDIX: Replication package

This appendix contains the data required to replicate our results.

Table 2: Originally estimated and eventual IRCC processing times

Participant ID	Originally estimated processing time	Eventual processing time
R-177-🎧👾942q🔋	119	190^{*+}

*and counting $^{+}$not anymore

References

[Blue, 2021] Blue, K. (2021). Towing capacity guide: Everything you need to know. https://www.kbb.com/car-advice/towing-capacity-guide/.

[Bovik, 1975] Bovik, H. (1975). The driving forces behind hyper driven devices. *Journal of Higher Spaces*, 1.

[Bovik and Fakir, 1988] Bovik, H. and Fakir, A. (1988). Hyper driven devices–theory and practice. *Journal of Higher Spaces*, 14:251–276.

[Bovik and Rude, 1986] Bovik, H. and Rude, B. (1986). Program vilification overview: 1970-1986. *Tourette's*, 7:97–124.

[CCOHS, 2022] CCOHS (2022). Space Requirements for Office Work. https://www.ccohs.ca/oshanswers/ergonomics/office/working_space.html.

[CHIRAG, 2020] CHIRAG (2020). How to Calculate the Gravitation Force from a Black Hole. https://public.nrao.edu/ask/how-to-calculate-the-gravitation-force-from-a-black-hole/.

[Einstein and Davis, 2013] Einstein, A. and Davis, F. A. (2013). *The principle of relativity.* Courier Corporation.

[Garrett-Bakelman et al., 2019] Garrett-Bakelman, F. E., Darshi, M., Green, S. J., Gur, R. C., Lin, L., Macias, B. R., McKenna, M. J., Meydan, C., Mishra, T., Nasrini, J., Piening, B. D., Rizzardi, L. F., Sharma, K., Siamwala, J. H., Taylor, L., Vitaterna, M. H., Afkarian, M., Afshinnekoo, E., Ahadi, S., Ambati, A., Arya, M., Bezdan, D., Callahan, C. M., Chen, S., Choi, A. M. K., Chlipala, G. E., Contrepois, K., Covington, M., Crucian, B. E., Vivo, I. D., Dinges, D. F., Ebert, D. J., Feinberg, J. I., Gandara, J. A., George, K. A., Goutsias, J., Grills, G. S., Hargens, A. R., Heer, M., Hillary, R. P., Hoofnagle, A. N., Hook, V. Y. H., Jenkinson, G., Jiang, P., Keshavarzian, A., Laurie, S. S., Lee-McMullen, B., Lumpkins, S. B., MacKay, M., Maienschein-Cline, M. G., Melnick, A. M., Moore, T. M., Nakahira, K., Patel, H. H., Pietrzyk, R., Rao, V., Saito, R., Salins, D. N., Schilling, J. M., Sears, D. D., Sheridan, C. K., Stenger, M. B., Tryggvadottir, R., Urban, A. E., Vaisar, T., Espen, B. V., Zhang, J., Ziegler, M. G., Zwart, S. R., Charles, J. B., Kundrot, C. E., Scott, G. B. I., Bailey, S. M., Basner, M., Feinberg, A. P., Lee, S. M. C., Mason, C. E., Mignot, E., Rana, B. K., Smith, S. M., Snyder, M. P., and Turek, F. W. (2019). The NASA twins study: A multidimensional analysis of a year-long human spaceflight. *Science*, 364(6436):eaau8650.

[Hawking, 2015] Hawking, S. W. (2015). The information paradox for black holes. *arXiv preprint arXiv:1509.01147*.

[Hrvatin, 2017] Hrvatin, V. (2017). Your canadian tire money might be priceless. https://www.macleans.ca/society/your-canadian-tire-money-might-be-priceless/.

[McKenzie and McKenzie, 1980] McKenzie, B. and McKenzie, D. (1980). The Great White North. SCTV.

[Nuseibeh and Easterbrook, 2000] Nuseibeh, B. and Easterbrook, S. (2000). Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 35–46.

[Pedia, 2022a] Pedia, W. (2022a). 2 Chainz. https://en.wikipedia.org/wiki/2_Chainz.

[Pedia, 2022b] Pedia, W. (2022b). Black holes. https://simple.wikipedia.org/wiki/Black_hole.

[Pedia, 2022c] Pedia, W. (2022c). Chain rule. https://en.wikipedia.org/wiki/Chain_rule.

[Pedia, 2022d] Pedia, W. (2022d). Gradient descent. https://en.wikipedia.org/wiki/Gradient_descent.

[Pedia, 2022e] Pedia, W. (2022e). Ja Rule. https://en.wikipedia.org/wiki/Ja_Rule.

[Pedia, 2022f] Pedia, W. (2022f). Time dilation. https://en.wikipedia.org/wiki/Time_dilation.

[Pedia, 2022g] Pedia, W. (2022g). Vim (text editor). https://en.wikipedia.org/wiki/Vim_(text_editor).

[Ralph, 2013] Ralph, P. (2013). The illusion of requirements in software development. *Requirements Engineering*, 18(3):293–296.

[Rittel and Webber, 1974] Rittel, H. W. and Webber, M. M. (1974). Wicked problems. *Man-made Futures*, 26(1):272–280.

[Sahay et al., 2020] Sahay, A., Indamutsa, A., Di Ruscio, D., and Pierantonio, A. (2020). Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 171–178. IEEE.

[San Fearlow and Ansymov, 2015] San Fearlow, M. and Ansymov, I. (E.U. Patent 9 876 543, Oct. 2015). Automated alarm system with humanoid audio interface for alerting me (and just me!) when Hans enters his office and I can finally access the coffee maker.

[Sansing, 2022] Sansing, S. (2022). 2021 Chevrolet Tahoe Overview. https://www.sandysansingchevrolet.com/research/new-chevy-tahoe-towing-capacity.

[Schrödinger, 1935] Schrödinger, E. (1935). Die gegenwärtige situation in der quantenmechanik. *Naturwissenschaften*, 23(50):844–849.

[Trottoir-Barré et al., 2016] Trottoir-Barré, É., von Pamplemousse, R., San Fearlow, M., and Ansymov, I. (2016). The Black Hole in Hans' Backpack: A Plausible Explanation of Delayed Meetings and Other Time-abusive Phenomena. *Metaversa Periodica*, 23:201–276.

[von Pamplemousse and Ansymov, 2019] von Pamplemousse, R. and Ansymov, I. (2019). Theoretically, if I move closer to Hans, I should be able to meet with him more regularly. *Transactions of Naïve Scientists*, 75:1–999.

[Zizek and Daly, 2004] Zizek, S. and Daly, G. (2004). Conversations with zizek.

Black Hole Computation

Matias Scharager

Carnegie Mellon University

mscharag@cs.cmu.edu

1 Introduction

Consider the universe we live in right now. We are all governed by several scientific laws of nature such as Newton's laws and many more. These laws create harmony in our universe, creating a natural system of order. We can say these laws restrict possible states of being as if the universe is being statically type-checked. We can measure states of being as we progress forwards in time in our universe, but there are also many things far beyond our comprehension. One of these things is potential parallel universes which can be hidden inside theoretical wormholes in black holes.[10]

All of this is simply a metaphor for programming languages. We start by defining the laws that govern the universe: the structural operational semantics. We can also prove the harmony and safety of these laws to show that our universe is stable in existence. Once this is achieved, we can write code in our language that type-checks, essentially the matter of our universe, and watch how it behaves as it executes.

In fact, every time we create a new programming language, we are defining a parallel universe![9] The way of reasoning in one language is different from the methodology of another language. This extends far beyond a simple syntax renaming: the information that can exist in the universe is altered due to the fundamental rules of the language.

In this paper, we take a look at the wormholes between programming language universes in an attempt to understand the interaction between these universes via logic and data transfer. In doing so, we uncover ingenious ways to perform computation, utilizing the computational power of alternate universes. This lens also allows understanding old ideas in a new way, suggesting alternative ways of creating compilers.

2 Compilers

Surprisingly, there is already a commonly used name for wormholes between universes: compilers. Compilers are structures that convert the logic and computational content in one language to that of another. As such, they act as a one-way ticket from a source to the target language.

Typically, compilers are considered separate entities that are not part of the programming language universe itself. The fact that we need to cross so many universe boundaries makes the compilation extremely inefficient in maintaining the expressive capacity of the program.

2.1 SML → C → TAL

Let's compile a Standard ML (SML)[4] program into Typed Assembly Language (TAL).[5] We write a compiler in C to do this conversion. If we think of each of these programming languages to be their own separate universe, then the C compiler exists in the C universe while the source code exists in the SML universe. The goal is to utilize the object of the C universe to teleport the SML code into the TAL universe.

We are capable of seeing the SML universe object in the C universe. This is done in a very clumsy way: through a syntax protocol. The SML code is expressed on paper via defining syntax arbitrarily for the different components of the language. This is then saved to a text file which is a rudimentary logic that

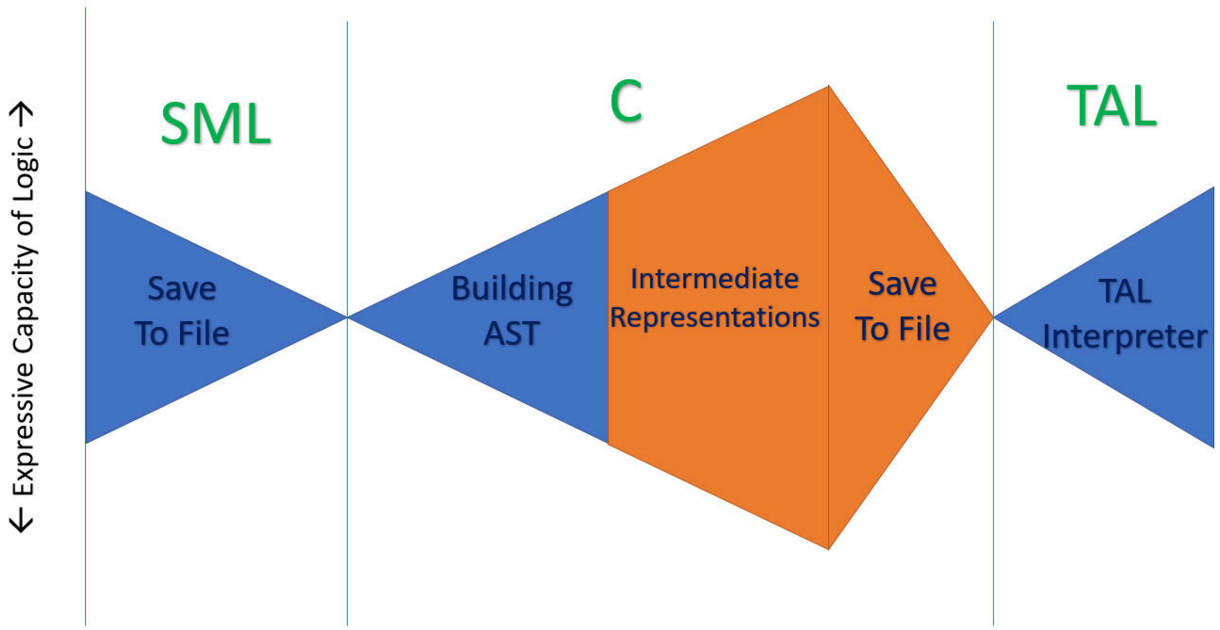

← Expressive Capacity of Logic →
SML
C
TAL
Save To File
Building AST
Intermediate Representations
Save To File
TAL Interpreter

Figure 1: Compilation Expressiveness

exists in all of the programming languages universes. Then the C compiler takes this file and operates on it via a parser. This is known as creating an abstract syntax tree (AST).

Consider the logical interpretation present at each stage of this transportation. We start in the SML universe with all the expressive power of SML, then we strip away all the expressiveness by converting it into a text file. This is a thinning down of the logic to make it easy to transfer. Upon entering the C universe, the compiler attempts to give back meaning to the text it is parsing, creating a simulated version of the SML universe to gain back some of the logic we had before. This thinning down and building back up of logic seems redundant.

We are not done here yet. We then create various intermediate representations and logics of the program we are compiling, defining various logical principles in the C universe. TAL expresses things in a different way than SML does, so we need to simulate all the changes in logic via multiple phases of compilation, increasing the logical interpretation of the program itself. Once all this is done, we save it to another file in the arbitrary syntax defined for TAL.

This out-file is transferred to a separate universe again, the TAL universe, where it can be executed in a comprehensible way on real-world computers. There can be further manipulations of this code as we need to increase our logical interpretation from the out-file to execution, but the details of this are beyond the scope of this paper.

The overall process can be seen in Figure 1. Notice that the expressiveness of the code at the beginning and at the end is roughly equivalent: we want our code to run exactly what we were planning on running right at the start. In between, there are a lot of inefficiencies where we lose expressiveness only to make an effort to gain it back up. Also, note the tightness of the informational content we can send through the barrier between universes. This is due to the only shared component between all these languages being a joint file system.

2.2 Removing First Boundary

The most direct attempt to salvage expressiveness across the program boundary is to implement the SML compiler in SML itself. If we naively attempt this, the result is that we still need to save the program to a file, destroying the expressive ability, only to convert it back into an AST to manipulate. Even if we simply converted the program into a “string” object instead of a file, we would still be losing informational content.

Instead, we must think in a more direct way. We want to convert our SML program into an SML AST of the same expressive content. The easiest conversion is the identity function: the SML program *is* an SML AST. Thankfully, the SML datatype system is a very nice way to write SML code in SML, and, moreover, the SMLNJ standard library includes an AST library! The library documentation can be found here: https://www.smlnj.org/doc/Compiler/pages/ast.html.

All SML programs that are normally written in SML syntax can alternatively be expressed in this AST framework with minimal impact on the code development experience. Moreover, we can easily convert already existing SML programs into this AST framework, allowing for forward compatibility.

To demonstrate the easiness of utilizing this AST library, we interviewed all the SML code developers in software engineering roles in the US. There were 0 complaints out of the large sample size of 0 SML code developers, providing empirical evidence for how easy it is for these engineers to convert to this representation.

If we consider the same “black hole” metaphor we utilize throughout the paper, our new compiler now exists in the same universe as the original source code, saving us the cost of sending the information through the black hole. We can think of this as the black hole sucking up the whole source code universe.

One of the major benefits of this conversion is blurring the distinction between programming and compilation. Every program must be self-compiling, meaning it must describe the appropriate operations needed to compile the program into a lower-level language.

Note that while we opted to discuss removing the C component of compilation, it is just as easy to base everything in C by using an SML AST library written in C to express our language. However, this would require coding in C, specifically writing an SML compiler in C, so it was instantly discarded as a bad idea.

2.2.1 Alternative Strategies

Since the current suggested framework for self compilation is rather complicated to implement, we can consider an extension to the SML language that includes an in-line compiler as a language feature. For example, here is the proposed hello world program.

```
COMPILE(
    print "Hello World!"
)
```

Notice that this COMPILE monad is admissible into our language, as we are capable of providing a computational interpretation of compilation. As such, we recover valid structural operational semantics without having to modify the type safety proofs of the language. As such, we are capable of expressing compilation as a program effect: it prints the compiled program to the output file.

There are several modifications we can make to this framework that would allow for greater user experiences. For one, we can allow for multiple kinds of COMPILE monads, representing which compiler flags we wish to have in different parts of the program. This will allow for a very hands-on approach to compiler optimizations, as we can mark which areas we want to have compiled with what strategies. We can even allow for a mix of the default COMPILE monads along with user implemented compilation so that we have the greatest amount of flexibility.

Once again, we are drawn back to our black hole analogy where monads represent universe teleportation. We are expressly marking which chunks of our current universe (SML) we want to send to an alternative universe (TAL).

2.2.2 Quines

It is worth mentioning the relationship of this format of programming to that of quines. A quine is any program that prints itself as its output [1]. As such, we can think of quines as a degenerate case of compilation where we do not perform any compilation steps, and simply spit back the original source code. Note that there is an additional concept of “multiquines” where compilation does in fact occur as one language is changed into another language [3], but ultimately, multiquines go back to the original source language after a finite number of steps, ruining the usefulness of the conversion.

2.3 Removing the Second Boundary

While we have now established a wonderful programming methodology to eliminate the first mistake of compilation, we have yet to explain how we wish to run the self-compiling program.

One option is to have a meta compiler, which compiles the self-compiling program. However, as Thompson suggests in Reflections on Trusting Trust[7], this is a very risky move, as the meta compiler could have been compiled by a meta meta compiler that has a trojan horse in it, essentially breaking the authenticity of our self compiling program. To make matters worse, if we choose to implement a meta compiler ourselves as a self compiling program, we would still need to write a meta meta compiler ourselves to compile the meta compiler, and we would end up writing an infinite hierarchy of compilers before we can safely run our program. This is essentially Russell's paradox in the form of compilation. It would be fair to say that there exists some magical black box compiler that exists as our initial self-compiling compiler, but such a thing would have to implement in an "engineering hack" kind of way in practice.

However, we are still in luck. The programming language we are expressing our program in has a valid small step relation, describing the process of executing. As such, we can conjecture the existence of a machine that would perform this small step relation as a morphism of our program, slowly progressing until we have reached a final stopping state at which we have achieved assembly output.

Unfortunately, such a perfect computer cannot exist in the real world with modern-day computer architecture, as it would require an infinite number of state transitions to model all possible whole program expressions a program can have during execution. As such, I would like you to take a brief moment of silence while reading this paper to lament the fact that our idealized computer is purely an ideal. Thank you for your moment of silence.

Let us take a step back and find out what exactly a computer in modern-day computer architecture is capable of computing. It is reasonable to claim that a computer can "run assembly code" meaning it is capable of performing operations that model the operations that assembly makes. The validity of this claim is left as an exercise to the reader. This means that we automatically have an assembly interpreter given to us magically.[8]

To remove the second boundary, we must implement an SML AST representation in an assembly program, then write an SML compiler in the assembly language to compile the SML ASTs that we write. Since everything is already in assembly, and we have a valid way of running assembly programs on a computer, then we have successfully preserved expressive capacity throughout the whole compilation process.

2.3.1 Syntactic Sugar

It is worth noting the definition of "syntactic sugar"[2] to assure ourselves that this compilation strategy for SML into assembly is not just syntactic sugar for assembly. Syntactic sugar is merely an admissible rule of computation being added into the program without defining a new structural operational semantics. In our setup, even though we are expressing SML ASTs in assembly, we are still distinguishing the structural operational semantics of SML from those of assembly, giving us a different set of expressiveness tools that can't be easily represented in assembly.

Note that under this definition of syntactic sugar, languages that do not have well-defined structural operational semantics can be considered to be just glorified assembly, since we claim that assembly is the only language we are capable of interpreting on a computer in practice. Aside from structural operational semantics, it seems sufficient to define a language via a specification such as C and thus a C to assembly compiler is a compiler, however the analytical tools we have for program analysis and compiler correctness would have to be vastly different in such a setting so it is beyond the scope of this paper.

3 Parallel Computation

If this were a sensible research paper, the term parallel computation would mean multi-threaded cores with fork and join operations, or even some parallelized dynamics formalization of the structural operational semantics. In our case, parallel computation means none of these things, yet at the same time means all of these things! Parallel is derived from the "parallel universes" that run concurrently to our current universe of existence. Parallel computation, therefore, means utilizing parallel universes for computation.

While our current universe is constrained to a certain set of logical principles, these principles do not necessarily apply to the parallel universe. As such, sending information to the parallel universe, and running the computations in that universe instead, allows us to save quite a lot of computation time among other things. We will now cover various useful parallel universes and their abilities.

3.1 Effect-Free Effects

It is often the case that we want to reason about the state of our environment. This might involve various operations including IO file reading, direct input from the user, reading and writing to memory, or utilizing cache or registers. However, these are always pesky to reason about from a programming language theory perspective.

We exclude non-termination in our definitions (we will cover this later separately), and define a pure function as one that does not reason about the state of the environment in any way. The advantage to this is the mathematical and category-theoretic idea that the function always returns the same thing, and is simply a mapping of input values to output values.

This means an impure function, otherwise known as an effectful function, is one that performs any of these IO operations. Clearly by the name "impure" we can see that this is an inferior version of pure functions. The advantage to it is we can do these "dirty" effectful computations, and this can save us computation time. For example, we all know from software engineering internship interviews that if we aren't sure how to make our solution faster, we yell out the words "hash set" or "dynamic programming" and hope that one of these two things is what the interviewer wants to hear.

We can go through the effort of making some of these operations pure but this usually requires a painstaking amount of monads and weird things that frequent the nightmares of software developers. Instead, we can simply create a universe where all these effects can exist, and so to calculate effectful functions, we send the data and run the program in this parallel universe. Since none of these effects occur in our current universe, we are never accessing the state of our environment, so such a function is considered pure.

With this, we recover all the niceties of pure functions within the current universe's structural operational semantics, and we handle the effects through a trivial lemma: since there are infinitely many parallel universes, there is one that has a nice behaving environment with reads and writes that we can send data to.

3.2 Infinity, Non-Termination, and Undecideability

There are several problems in computer science that are very important that we wish to compute but haven't found the means to. For example, some very important research questions include "what is the biggest natural number," "what is the last digit of pi," and "does this arbitrary Turing machine terminate on this input."

It is trivial to write a function that computes the largest natural number utilizing continuation-passing style:

```
datatype n = Z | S of n
open SMLofNJ.Cont

fun biggest () =
    let
        fun big (f : n -> n) = big (fn x => f (S x))
    in
        callcc (fn k => big (throw k))
    end

val ans = biggest()
```

Unfortunately, executing this program results in non-termination as this function infinitely loops. We would ideally like to keep running this program until it terminates, but since our universe is constrained to running things in a finite amount of time, we will never find the answer we want to find.

However, parallel universes are not constrained in the same way as our universe. We can find a parallel universe that runs an infinite number of steps of the program instantaneously and then send this program to be run in that universe. In such a universe, the concept of the last natural number can be made a reality, along with the last digit of pi, so we will refer to this universe as Infinity Universe.

We all know that the last question we asked is the famous Halting Problem. Fundamentally, this problem is undecideable, which means that no algorithm can provide a yes or no solution to it. How then can we correctly state that an algorithm in Infinity Universe gives us the answer to an undecideable problem? For this, we need a simple lemma: Infinity Universe allows for paradoxes. If time can be run infinitely instantaneously, then if we consider the current time that we are in as our starting point, then this is no different of a starting point than one second into the future, or a year, or a century, or even a larger time gap like the length of time it takes mankind to eradicate covid. This means that current time + 1 unit of time = current time. If we standardize our current time to being time zero, this equation shows us that $1 = 0$. Previous research [6] demonstrates a computer verified proof of arbitrary program termination in polynomial time in an unsound type system, solving the P=NP problem. In fact, we can prove any program halts, as we can simply run it in Infinite Universe, demonstrating that Infintite Universe is simply an unsound universe.

A quick note, I would rather not wish to exist in an unsound universe. I am uncertain as to what that would do to the basic laws of physics, but it is worth conducting experiments to determine this.

3.3 Implementing Parallel Universes

As we just discussed the extreme usefulness of these parallel universes, we would like a way of modeling them in our programming language. Again, we consider the black hole principle. We are currently in universe A which we know and comprehend (have structural operational semantics). We wish to run a program in universe B, where we know nothing about how things run. So we open up a black hole, and send the program into the black hole!

This black hole is a monad that allows for communication from universe A to universe B. We can define some construct DATA (or abbreviated E) that allows us to express a syntactic formulation of contents in universe B expressible in universe A. We can also define bind and return operations to express the monadic structure. Here is an example of how data might look like if universe B was a universe filled with cats.

$$\begin{aligned} M &:= x \mid () \mid \langle M, M \rangle \mid M \cdot 1 \mid M \cdot 2 \mid \lambda x : A.M \mid M\ M \mid \texttt{blackhole}(E) \\ E &:= \texttt{ret}(M) \mid \texttt{bnd}(M; x.E) \mid \texttt{MEOW} \mid \texttt{PURR} \mid \texttt{NYA} \mid E\ \texttt{owo}\ E \end{aligned}$$

We can now express the program that explains the command to solve the Halting Problem in universe B.

$$\texttt{blackhole(NYA owo MEOW)} : \bigcirc(\texttt{HALTING PROBLEM})$$

As we can see, this is a valid term of the type $\bigcirc(\texttt{HALTING PROBLEM})$. It provides to us a function that solves the Halting Problem expressed in universe B.

We now witness a dilemma. How can this possibly be true? Well, universe B is not constrained to any meaningful principles that universe A obeys. However, the backlash to this is that no information in universe B is comprehensible in universe A. We can continue to transmit information to universe B from universe A via this black hole and continue to compute things like solving the Halting Problem in universe B, but the black hole is a one-way street. Once you've gone to a universe of cats, you will never want to leave.

However, this allows for quite some grand amount of interesting properties in the universe A. Since all of these $\texttt{blackhole}(E)$ terms reduce within the universe B instead of universe A, and none of the information is observable within universe A, then it is accurate to state that for all E, E', $\texttt{blackhole}(E) \simeq \texttt{blackhole}(E')$ where $\simeq$ represents Kleene equivalence over any observation, or can be expanded to be contextual equivalence. As such, we can recover the structural operational semantics of the term language M in universe A via the following reduction

$$\frac{}{\texttt{blackhole}(E) \mapsto \texttt{blackhole}(\bullet)} \qquad \frac{}{\texttt{blackhole}(\bullet)\ \texttt{val}} \qquad \frac{}{\Gamma \vdash \texttt{blackhole}(\bullet) : \bigcirc(A)}$$

Where $\bullet$ represents the fact that E was successfully transmitted to universe B. As such, progress and preservation are simple inductions on the static and dynamic rules, and we assure type safety.

3.3.1 Parallel Parallel Universes

There is no reason to limit ourselves to simply two universes as we have in our previous example. We can have infinitely many parallel universes, and continue to expand our language constructs via adding these extra parallel universes.

In fact, we every time we add an extra monad into our system, we are only making moderate corrections to the proof of type safety, along with all the logical relations associated with the programming language. This allows for an easy method of extending our language.

3.4 Recovering the Other-Worldly Data

Even though data sent through a black hole to a separate universe can never come back, it is still the case that something might come back to our universe if there is a black hole in the other universe connecting it to ours. As such, this suggests that a bidirectional message passing system between universes is possible.

It is interesting to draw ties from this idea to that of the π-calculus. For the most part, we consider the case where processes in the π-calculus correspond to the same universe so we can consider communication via channels in this language as black holes which simply return you to the same universe you started from. This restriction doesn't seem to be necessary, as different processes can utilize different operational semantics without breaking the abstractions of the π-calculus.

Constructing a system for message passing between multiple universes is beyond the scope of this paper, because it's already hard enough to implement the structural operational semantics for one language, let alone two or more. If we intend to pass messages between universes, we can no longer just hypothetically suppose the operations of one language existing, but must actually provide it in order to run the program.

4 Conclusions

As we discovered in this paper, we can implement a compilation strategy that preserves the expressiveness of the program logic throughout the whole compilation process. This is done via expressing every program we write as a self-compiling, self-interpreting program in the assembly language.

We also discovered a methodology for performing any arbitrary effectful or undecidable computation in constant time by transmitting the information content to a separate universe via a black hole monad. This paves the way for several interesting implementations in parallel universes that are observable via our own.

References

[1] Douglas R. Hofstadter. *Godel, escher, bach.* Basic Books, 1979.

[2] Peter J Landin. The mechanical evaluation of expressions. *The computer journal*, 6(4):308–320, 1964.

[3] David Madore. Quines (self-replicating programs). `http://www.madore.org/~david/computers/quine.html`.

[4] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The definition of standard ML: revised.* MIT press, 1997.

[5] Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From system f to typed assembly language. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 21(3):527–568, 1999.

[6] Matias Scharager. Verified proof of p=np in the silence theorem prover language. In *SIGBOVIK 2020*, SIGBOVIK, pages 51–54, Pittsburgh, PA, 2020. Association for Computational Heresy.

[7] Ken Thompson. Reflections on trusting trust. In *ACM Turing award lectures*, page 1983. 2007.

[8] Wikipedia. Incantation. `https://en.wikipedia.org/wiki/Incantation`.

[9] Wikipedia. Isekai. `https://en.wikipedia.org/wiki/Isekai`.

[10] Wikipedia. Wormhole. `https://en.wikipedia.org/wiki/Wormhole`.

Schrödinger's SAT: Generalizing Quantum Bogosort to Prove P = NP Under Many-Worlds Quantum Mechanics

Melody "`boringcactus`" Horn

ABSTRACT

Quantum bogosort is a well-known variant of bogosort that exploits the quantum nature of the universe to sort a list in linear time under the many-worlds interpretation of quantum mechanics. We generalize this algorithm to solve the Boolean satisfiability problem in $O(n)$ time. The Boolean satisfiability problem is the original NP-complete problem; as such, this proves that P = NP. This destroyes the RSA cryptosystem.

KEYWORDS

satisfiability, time complexity, revolutions in our understanding of computing, unsolved problems in millennium prize eligibility

1 INTRODUCTION

The Boolean satisfiability problem was the first problem proved to be NP-complete [1, 4]. This result serves as the foundation for all of complexity theory.

Bogosort is a randomized list sorting algorithm that runs in average-case $O(n!)$ time [2]. Quantum bogosort is an adaptation of bogosort that explores all random options simultaneously in different universes and therefore sorts the list in $O(n)$ time [3].

2 PRIOR ART

The bogosort algorithm given in [2] sorts an array a with n elements as follows:

Algorithm 1 Bogosort

```
1: procedure BOGOSORT($a$)
2:     while $a[1 \dots n]$ is not sorted do
3:         randomly permute $a[1 \dots n]$
4:     end while
5: end procedure
```

The quantum bogosort algorithm given in [3] may be formalized analogously as follows:

Algorithm 2 Quantum Bogosort

```
1: procedure QUANTUM-BOGOSORT($a$)
2:     randomly permute $a[1 \dots n]$
3:     if $a[1 \dots n]$ is not sorted then
4:         destroy the entire universe
5:     end if
6: end procedure
```

So long as the random numbers in step 2 are random at a quantum level, the many-worlds interpretation of quantum mechanics indicates that there will be a world where each random permutation is chosen. As such, step 4 ensures that only the worlds where the correct random permutation was chosen continue to exist. Since steps 2 and 3 can run in $O(n)$ time, and step 4 is independent of n and therefore runs in $O(1)$ time, quantum bogosort will sort the array a in $O(n)$ time.

3 METHODS

The Boolean satisfiability problem can be formalized as follows: given some Boolean formula $\Phi(x_1, \dots, x_n)$ on n variables, find a truth assignment $(x_1, \dots, x_n) = (T, \dots, F)$ such that $\Phi(x_1, \dots, x_n)$ is true, if it exists.[1]

To solve this problem, we present the following algorithm:

Algorithm 3 Schrödinger's SAT

```
1: procedure SCHRÖDINGER'S-SAT($\Phi$)
2:     for $i \leftarrow 1, n$ do
3:         Randomly guess either $x_i \leftarrow T$ or $x_i \leftarrow F$
4:     end for
5:     if $\neg\Phi(x_1, \dots, x_n)$ then
6:         destroy the entire universe
7:     end if
8:     return $(x_1, \dots, x_n)$
9: end procedure
```

As with quantum bogosort, if the guess in step 3 is random at a quantum level, there will be a world for each value, and so by step 4 there is a world for every possible truth assignment. As such, by step 8 we have found a satisfying truth assignment. (Sufficiently bored or curious readers may wish to implement this algorithm and run it on $\Phi(x_1) = x_1 \wedge \neg x_1$.)

Since there are n guesses made, each guess takes $O(1)$ time, and the formula can be evaluated in $O(n)$ time, this algorithm finds a satisfying assignment in $O(n)$ time, demonstrating that Boolean satisfiability is in P and therefore that P = NP.

ACKNOWLEDGMENTS

To Joe.

REFERENCES

[1] Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*. ACM Press, Shaker Heights, Ohio, United States, 151–158. https://sci-hub.se/10.1145/800157.805047

[2] Hermann Gruber, Markus Holzer, and Oliver Ruepp. 2007. Sorting the Slow Way: An Analysis of Perversely Awful Randomized Sorting Algorithms. In *Fun with Algorithms*, Pierluigi Crescenzi, Giuseppe Prencipe, and Geppino Pucci (Eds.). Vol. 4475. Springer Berlin Heidelberg, Berlin, Heidelberg, 183–197. https://sci-hub.se/10.1007/978-3-540-72914-3_17 Series Title: Lecture Notes in Computer Science.

[3] The Other Tree. 2009. Quantum Bogosort. *mathNEWS* 111, 3 (Oct. 2009), 13. https://mathnews.uwaterloo.ca/wp-content/uploads/2014/08/v111i3-compressed.pdf

[1] As is standard practice, we handwave away the difference between the decision problem and the search problem.

[4] B.A. Trakhtenbrot. 1984. A Survey of Russian Approaches to Perebor (Brute-Force Searches) Algorithms. *IEEE Annals of the History of Computing* 6, 4 (Oct. 1984), 384–400. https://sci-hub.se/10.1109/MAHC.1984.10036

Solving Double Execution of Java's `paint()` Method by Counting Down to the Heat Death of the Universe (plus language compendium)

Braden Oh, Vedaant Kuchhal, Junseok Kang, Andrew Mascillaro, Justin Kunimune, Shashank Swaminathan, Devlin Ih, Elias Gabriel, Audrey Lee, Colin Snow, Hyunkyung Rho, Ben Morris, Solomon Greenberg, Mahima Beltur, Anusha Datar, Jonathan Kelley, Pranavi Boyalakuntla, Xander Hughes, Devyn Oh, Aidan Schmitigal

Olin College of Engineering: Computing

Contents

1 Background and Motivation

2 Results

3 Language Compendium

3.1 Java . . . .
3.2 Python 3 . . . .
3.3 B . . . .
3.4 C . . . .
3.5 D . . . .
3.6 JavaScript . . . .
3.7 TypeScript . . . .
3.8 MATLAB . . . .
3.9 Ruby . . . .
3.10 Elisp . . . .
3.11 Common Lisp . . . .
3.12 Go . . . .
3.13 Racket . . . .
3.14 Scratch . . . .
3.15 OCaml . . . .
3.16 Bash . . . .
3.17 Fortran . . . .
3.18 ArnoldC . . . .
3.19 LOLCODE . . . .
3.20 PHP . . . .
3.21 Verilog . . . .
3.22 R . . . .
3.23 Swift . . . .
3.24 Dyalog APL . . . .
3.25 COBOL . . . .
3.26 Mathematica . . . .
3.27 Perl . . . .
3.28 Lua . . . .
3.29 Julia . . . .
3.30 Rust - Looping . . . .
3.31 Rust - Arbitrary Precision . . . .
3.32 MIPS Assembly . . . .
3.33 NetFuck . . . .
3.34 brainfuck . . . .
3.35 Befunge-98 . . . .
3.36 Minecraft Redstone . . . .

1 Background and Motivation

The final project of my high school "Honors C++" course was to program a graphical animation in a Java applet. While writing this program I found that the animation would consistently run twice in a row, meaning that the pre-programmed animation would run all the way through then would immediately restart, preventing the user from being able to either recognize or appreciate the ending of the cinematic masterpiece as they were forced to watch the film a second time.

Investigation of this bug led me to the `paint()` method employed by the Java Virtual Machine's (JVM) graphics engine. The JVM calls the `paint()` method to generate the graphics displayed in a window. This method is called automatically anytime the JVM determines that the graphical user interface (GUI) needs to be refreshed. Ideally this occurs only when the GUI actually does need to be refreshed, but, in practice, the JVM will execute the paint method when no reason for a GUI refresh is apparent. The JVM's double execution of `paint()` command is a symptom described on numerous message boards including Stack Overflow, MacRumors, Code Ranch, and Tek Tips. Unfortunately the JVM's reason for re-executing the `paint()` method is opaque to the programmer, so the actual cause of the double execution could not be easily investigated. Thus, in order to prevent the user from becoming confused, a clever programmatic solution would be required. Because the program was only an animation that did not require any responsiveness to user input, an easy way to make the double execution invisible to the user was to insert a block of code at the end of the animation, right at the end of the `paint()` method, to consume an enormous amount of time, giving the user more time to view the final image displayed in the window. After implementing enormous addition problems which produced only seconds of delay I began to experience a burning desire to defeat the JVM and guarantee that it could never run my animation again.

What better way to assure this than to have the `paint()` method wait until the heat death of the universe? This would guarantee that the immutable laws of thermodynamics would step in to prevent the JVM from executing a second time, even in the event that the computer remained switched on and plugged into a consistent power source for billions of years. The Wikipedia page entitled "Heat death of the universe" reports that the Hawking radiation evaporation of a supermassive black hole of 10^{11} Earth masses is on the order of 10^{100} years. This amount of time seemed sufficient to prevent `paint()` from ever running again, so I wrote a simple Java function to count down that many seconds and called it at the end of my implementation of the `paint()` function.

2 Results

I am happy to report that this solution worked perfectly. The `paint()` function executed for the first time and then began counting down in the background. My apparent fix to the double execution of `paint()` impressed my programming teacher, but also confused him as he saw the script continued to run beyond the animation's apparent conclusion. He terminated the program's operation prior to `paint()`'s second execution, rendering this solution a practical success.

This algorithm of counting down to the heat death of the universe provides a valuable solution to any programmer needing to assure that no further computations can take place following a particular block of code. In order to provide as universal a solution as possible, we have provided translations of this algorithm in the only programming languages that matter.

3 Language Compendium

3.1 Java

The maximum 32-bit integer value that Java can store is 2,147,483,647, which is on the order of 10^9. To make the math easy, I used the integer 1×10^9 as a loop counter, then looped through 10^{100} years, Calling `Thread.sleep(1000)` (1000 ms) to count through each of the 31,536,000 seconds in a year.

```
import java.lang.Thread;
public void wait_for_heat_death_of_the_universe() {
    // Loop through 10^100 years
    for(long count1 = 0; count1 <= 1000000000; count1++) {
    for(long count2 = 0; count2 <= 1000000000; count2++) {
    for(long count3 = 0; count3 <= 1000000000; count3++) {
    for(long count4 = 0; count4 <= 1000000000; count4++) {
    for(long count5 = 0; count5 <= 1000000000; count5++) {
    for(long count6 = 0; count6 <= 1000000000; count6++) {
    for(long count7 = 0; count7 <= 1000000000; count7++) {
    for(long count8 = 0; count8 <= 1000000000; count8++) {
    for(long count9 = 0; count9 <= 1000000000; count9++) {
    for(long count10 = 0; count10 <= 1000000000; count10++) {
    for(long count11 = 0; count11 <= 1000000000; count11++) {
    // Loop through 31,536,000 seconds per year
    for(long count12 = 0; count12 <= 31536000; count12++) {
        try{Thread.sleep(1000);}    // Wait 1000 ms
        catch(InterruptedException ie) {}  // This is a syntactical formality
    }}}}}}}}}}}}
}
```

3.2 Python 3

Astoundingly, Python 3 allows a user to handle the number `1e100`, which is represented as a floating-point number. Even more astoundingly, Python even allows a user to convert it into an integer which can be used as a loop counter (for reference, the `sys.getsizeof` method indicates that this integer requires 72 bytes of memory). This number is actually represented in memory as a number slightly larger than `1e100`, but the difference is mere fractions of a percent (albeit that doesn't mean much at numbers of this order of magnitude).

```
import time
def wait_for_heat_death_of_the_universe():
    for count in range(0, int(1e100)):  # Loop through 10^100 years
        time.sleep(31536000)            # Wait for one year (in seconds)
```

3.3 B

B is a predecessor of C written by B. W. Kernighan at Bell Labs. It does not implement types, for loops, or sleep commands, but does support while loops and recursion as well as system calls. Here we recursively call the UNIX "sleep 31536000" command 10^{100} times to achieve our desired time.

```
main() {
    WaitForHeatDeath(100);
}

WaitForHeatDeath(n) {
    if (n == 0) {
        system("sleep 31536000");
        return;
    }
    auto a, b;
    a = 10;
    b = 0;
    while(a > b) {
        b = b + 1;
        WaitForHeatDeath(n - 1);
    }
}
```

3.4 C

This is a very similar implementation to that of Java, except for one key difference - the `long` type in C can be 32 or 64 bit depending on the operating system, so a 64-bit integer - using the standard integer library - was specified, and for loops were nested accordingly with a one year sleep function from the Unix standard library.

```
#include <unistd.h>
#include <stdint.h>
int main(){
    for(int64_t count1 = 0; count1 <= 10000000000; count1++){
    for(int64_t count2 = 0; count2 <= 10000000000; count2++){
    for(int64_t count3 = 0; count3 <= 10000000000; count3++){
    for(int64_t count4 = 0; count4 <= 10000000000; count4++){
    for(int64_t count5 = 0; count5 <= 10000000000; count5++){
    for(int64_t count6 = 0; count6 <= 10000000000; count6++){
    for(int64_t count7 = 0; count7 <= 10000000000; count7++){
    for(int64_t count8 = 0; count8 <= 10000000000; count8++){
    for(int64_t count9 = 0; count9 <= 10000000000; count9++){
    for(int64_t count10 = 0; count10 <= 10000000000; count10++){
        sleep(31536000);
    }}}}}}}}}}
}
```

3.5 D

D is a general-purpose programming language with static typing, systems-level access, and C-like syntax. Thus, the implementation is effectively the same as C, noting the different modules.

```
import std;
import core.thread;
void main(){
    for(int64_t count1 = 0; count1 <= 10000000000; count1++){
    for(int64_t count2 = 0; count2 <= 10000000000; count2++){
    for(int64_t count3 = 0; count3 <= 10000000000; count3++){
    for(int64_t count4 = 0; count4 <= 10000000000; count4++){
    for(int64_t count5 = 0; count5 <= 10000000000; count5++){
    for(int64_t count6 = 0; count6 <= 10000000000; count6++){
    for(int64_t count7 = 0; count7 <= 10000000000; count7++){
    for(int64_t count8 = 0; count8 <= 10000000000; count8++){
    for(int64_t count9 = 0; count9 <= 10000000000; count9++){
    for(int64_t count10 = 0; count10 <= 10000000000; count10++){
        Thread.sleep( dur!("seconds")( 31536000 ) );
    }}}}}}}}}}
}
```

3.6 JavaScript

With the new update of JavaScript, there is no need to consider the Number data type's MAX_SAFE_INTEGER. The BigInt Datatype allows a way to represent whole numbers larger than 2^{53}-1. With this, we can use a similar structure to Python's for loop and create this function.

```
function end_of_time(){
    for (let i = BigInt(0); i < BigInt(1e100); i++) {
        sleep(31536000);
    }
}
```

3.7 TypeScript

Since the JavaScript implementation is nearly unreadable, TypeScript fortunately allows us to declare the type of each variable in the source code, which is desirable for any large and complicated program like this one.

```
function end_of_time(): void {
    for (let i: number = BigInt(0); i < BigInt(1e100); i++) {
        sleep(31536000);
    }
}
```

3.8 MATLAB

This implementation is as simple as it can get - initially, it was tempting to iterate across an array from 1 to 1e100 (it's difficult to find exact data on MATLAB's maximum array size), but an array of that size would well exceed the maximum size that any PC can handle. So looping through 10^{10} 10 times proved to be an effective solution. MATLAB provides the inbuilt `pause` function to pause each iteration for one year.

```
function wait_for_heat_death_of_the_universe()
    % Loop through 10^100 years
    for year1 = 1:10000000000
        for year2 = 1:10000000000
            for year3 = 1:10000000000
                for year4 = 1:10000000000
                    for year5 = 1:10000000000
                        for year6 = 1:10000000000
                            for year7 = 1:10000000000
                                for year8 = 1:10000000000
                                    for year9 = 1:10000000000
                                        for year10 = 1:10000000000
                                            % Pause for one year
                                            pause(31536000);
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
```

3.9 Ruby

As with Python, Ruby allows for the definition of arbitrarily sized integers. At scales larger than a machine word, where a buffer overflow may occur, Ruby will automatically convert any integer to a bignum representation. In this representation, integers consume multiple bytes. The number 10^{100} consumes only 42 bytes (compared to the 72 used by Python), making it a better-suited implementation choice for those concerned with memory usage.

Unfortunately, Ruby's `sleep` implementation can only act on integers within machine precision, so though our desired amount of time `(10 ** 100) * 31536000` can be represented, it cannot be used. Fortunately, as with other methods, we can get around this by instead delaying for 1 year 10^{100} times.

```
(10 ** 100).times { sleep 31536000 }
```

3.10 Elisp

Emacs Lisp, or Elisp, is the extension language for the GNU/Emacs text editor. It is probably the most commonly used dialect of the Lisp family.

GNU/Emacs users tend to constantly promote their silly editor from the 1980s, and how it is better than modern IDEs (me). Sometimes you want to tell those people to shut up. You can lock their single-threaded editor until the heat death of the universe by evaluating the following snippet. Hilariously, ELisp, a language meant primarily for text manipulation and writing editor commands, allows arbitrary precision integers with its bignum type.

```
(dotimes (count (expt 10 100))
  (sleep-for 31536000))
```

3.11 Common Lisp

Common Lisp is a powerful, general purpose, multi-paradigm programming language in the Lisp family. CL is known for its stability, speed, powerful macro system based on s-expression manipulation, and it's unparalleled interactive programming experience through the SLIME environment.

CL supports arbitrary precision integers through the bignum type, which will let you loop over 10^{100} times.

The following code should be implementation agnostic, and is currently being tested with SBCL.

```
(dotimes (count (expt 10 100))
  (sleep 31536000))
```

3.12 Go

The Go implementation follows a process similar to both JavaScript and the more legible TypeScript alternative. Sadly, Go is more verbose in defining big integers, as standard operators such as `**`, `<`, and `+` are not overloaded. Rather, function calls to perform said operations are used (`big.Int.Exp`, `big.Int.Cmp`, `big.Int.Add` respectively). As with Ruby, this implementation consumes 42 bytes (333 bits), but is likely more efficient as it is compiled rather than interpreted.

```
import (
    "time"
    "math/big"
)

one := big.NewInt(1)
eons := new(big.Int).Exp(big.NewInt(10), big.NewInt(100), nil)
counter := big.NewInt(0)

for ; counter.Cmp(eons) == -1; counter.Add(counter, one) {
    time.Sleep(31536000 * time.Second)
}
```

3.13 Racket

Racket is a member of the Lisp family of programming languages and is a descendent of Scheme. It is a language designed for defining domain specific languages, most well known for “teaching languages” used to teach functional programming concepts.

Racket, like the other lisps included here, supports arbitrary precision integers with bignums.

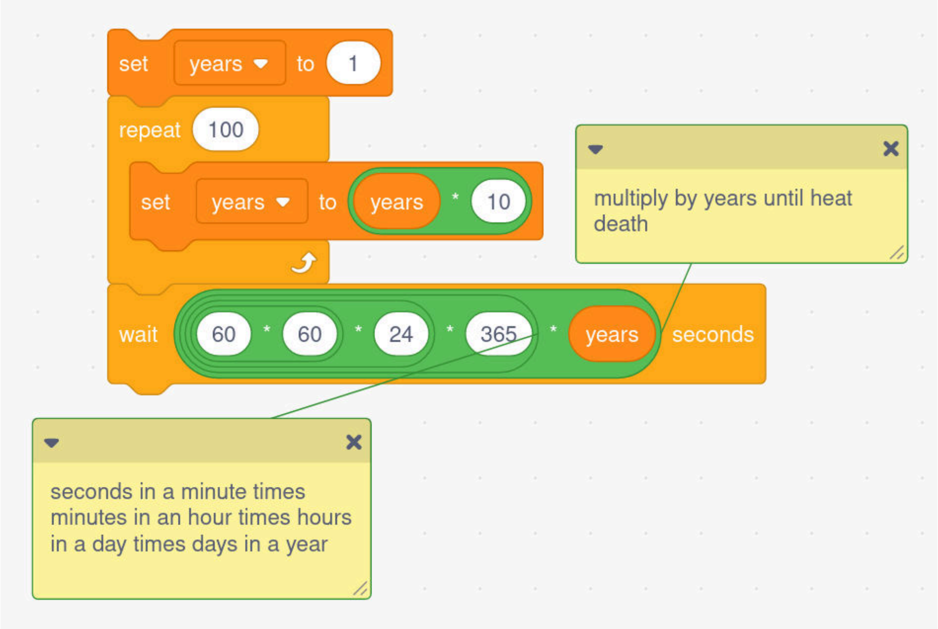
```
#lang racket

(define (heat-death year)
  (when (< year (expt 10 100))
    (sleep 31536000)
    (heat-death (add1 year))))

(heat-death 0)
```

3.14 Scratch

Scratch is a visual block-based programming language and game engine. It is commonly used in schools and classrooms to introduce young children to core programming concepts. As such, it may be useful to utilize when teaching concepts such as the inevitability of universal heat death, and its applicability to painting. See Figure 1 for implementation.



```
set years to 1
repeat 100
  set years to years * 10
wait 60 * 60 * 24 * 365 * years seconds
```

multiply by years until heat death

seconds in a minute times minutes in an hour times hours in a day times days in a year

Figure 1: In the code above, comments are provided for visual comprehension.

3.15 OCaml

OCaml is a powerful statically functional programming language, commonly used to teach core computer science topics. It’s portability, speed, and expressiveness make it an ideal candidate for development, thus a situation in which one would want to pause execution of a misbehaving program is likely to occur. Tragically, OCaml removed built-in support for arbitrarily-sized integers in recent versions, offloading that functionality to a separate packaged `zarith`.

Installing that package, we can make use of it in a similar way as above (that is, instructing your program to wait for 31536000 seconds 10^{100} times). On a 64bit architecture, OCaml uses 48 bytes to store our eon counter.

```
#load "zarith.cma";;

let rec wait_heat_death (remaining: Z.t): bool =
    if remaining = Z.zero then true else
        let _ = Unix.sleep 31536000 in wait_heat_death (Z.pred remaining);;

let _ = wait_heat_death (Z.pow (Z.of_int 10) 100);;
```

3.16 Bash

Since Bash coerces string data into integers whenever necessary, the best way to define the value 10^{100} without floating point errors is to create a string of 1 followed by 100 zeroes, as done in the first line below. Then, iterate over each of these years and sleep for 365 days.

```
years="1"$(printf "0%.0s" {1..100})
for i in seq $years; do sleep 365d; done
```

3.17 Fortran

Algorithmically, the Fortran implementation is similar to all other nested loop-based approaches.

```
Program HeatDeath
do a = 0, 1000000000
do b = 0, 1000000000
do c = 0, 1000000000
do d = 0, 1000000000
do e = 0, 1000000000
do f = 0, 1000000000
do g = 0, 1000000000
do h = 0, 1000000000
do i = 0, 1000000000
do j = 0, 1000000000
call sleep(31536000)
end do
end do
end do
end do
end do
end do
end do
end do
end do
end do
End Program HeatDeath
```

3.18 ArnoldC

ArnoldC is an imperative JVM-bytecode assembly language in which instructions and basic keywords are replaced with one-liners from different Arnold Schwarzenegger movies. The language, having its syntax comprised of well-known phrases, is an ideal choice for applications developed by fans and movie-goers. Promisingly, its code is also readable by almost any individual, with or without any formal programming knowledge, as it only requires proficiency in English and Schwarzenegger mannerisms. The language, compiling directly to JVM bytecode, is also able to run on any platform without recompilation.

Tragically, ArnoldC has no built-in support for bignums, timing functionality, nor any way to make system calls. As such, traditional methods like those presented in other languages are provably impossible.

Building on the work done by Lambert and Power[1], however, we can approximate a solution using the execution time of a known bytecode. As shown in their paper, we know with confidence that any used bytecode time is platform-independent (in every regard except processor speed, for which we account during usage), making it an useful drop-in as a base unit of elapsing time in a general-purpose solution. From the 137 timed bytecodes, we pick integer division (`idiv`) due to its comparatively (by an order of magnitude) longer execution time of 3.449739×10^{-8} seconds vs. all other used operations; thus, our implementation can delay for longer increments and be more efficient. Floating point conversions (`d2i` and `f2i`) take longer, but cannot be used as ArnoldC only supports 16-bit signed integers.

Combining this novel unit of time with the recursive approach demonstrated in B and the common looping approach, and knowing that our selected bytecode consumes a magnitude more time than the other programmatic instructions, we can successfully implement an ArnoldC program that will amortizedly delay for our desired time. To achieve heat death, we must recurse the number of times it takes to reach 10^{100} years, which assuming a time unit of 3.449739×10^{-8} seconds, equates to $10^{100} * \frac{31536000}{3.449739 \times 10^{-8}} \approx 10^{116}$ iterations. The implementation provided achieves this through deep recursion, where each recurse recurses 10 times.

Lambert and Power calculated the used bytecode instruction time using a ≈1GHz dual core Intel Pentium III. To account for processors operating outside of the 1-10GHz range, the programmer can simply decrement or increment the magnitude of iterations for each magnitude difference in processor speeds ($116 \pm 1n$).

[1] Lambert, J. M., J. F. Power, *Platform Independent Timing of Java Virtual Machine Bytecode Instructions*, Electronic Notes in Theoretical Computer Science. **220** (2008), pp. 97–113.

```
LISTEN TO ME VERY CAREFULLY HEATDEATH
I NEED YOUR CLOTHES YOUR BOOTS AND YOUR MOTORCYCLE n
    HEY CHRISTMAS TREE flag
    YOU SET US UP @I LIED
    GET TO THE CHOPPER flag
        HERE IS MY INVITATION n
        YOU ARE NOT YOU YOU ARE ME @I LIED
    ENOUGH TALK
    BECAUSE I'M GOING TO SAY PLEASE flag
        DO IT NOW WAITOPCODE
    BULLSHIT
        GET TO THE CHOPPER n
            HERE IS MY INVITATION n
            GET DOWN 1
        ENOUGH TALK
        DO IT NOW WAITMULTI n
    YOU HAVE NO RESPECT FOR LOGIC
HASTA LA VISTA, BABY
LISTEN TO ME VERY CAREFULLY WAITMULTI
I NEED YOUR CLOTHES YOUR BOOTS AND YOUR MOTORCYCLE n
    HEY CHRISTMAS TREE isLessThan10
    YOU SET US UP @NO PROBLEMO
    HEY CHRISTMAS TREE i
    YOU SET US UP @I LIED

    STICK AROUND isLessThan10
        GET TO THE CHOPPER i
            HERE IS MY INVITATION i
            GET UP 1
        ENOUGH TALK
        DO IT NOW HEATDEATH n
        GET TO THE CHOPPER isLessThan10
            HERE IS MY INVITATION 10
            LET OFF SOME STEAM BENNET i
        ENOUGH TALK
    CHILL
HASTA LA VISTA, BABY
LISTEN TO ME VERY CAREFULLY WAITOPCODE
    HEY CHRISTMAS TREE pi
    YOU SET US UP @I LIED
    GET TO THE CHOPPER pi
        HERE IS MY INVITATION 355
        HE HAD TO SPLIT 113
    ENOUGH TALK
HASTA LA VISTA, BABY
IT'S SHOWTIME
    DO IT NOW HEATDEATH 116
YOU HAVE BEEN TERMINATED
```

3.19 LOLCODE

Unfortunately, LOLCODE 1.2 (the most recent specification, released in 2007) does not yet support a time-based pause function such as Java's `Thread.sleep` or Python's `time.sleep`, nor does it support allowing a user to access the clock of the host computer. As a result, the original algorithm cannot be implemented directly in LOLCODE. LOLCODE is extremely similar to Arnold C in capability, but due to the short time available to this author, a similar outcome was achieved with an infinite loop.

```
HAI 1.2
CAN HAS STDIO?
HOW IZ I WAIT_4_HEAT_DTH_OF_UNVRSE
    VISIBLE "COUNTING DOWN"
    IM IN YR LOOP UPPIN YR VAR WILE WIN
        VISIBLE VAR
    IM OUTTA YR LOOP
IF U SAY SO
I IZ WAIT_4_HEAT_DTH_OF_UNVRSE MKAY
KTHXBYE
```

3.20 PHP

PHP, or Personal Home Page, or PHP Hypertext Protocol, is a programming language that was originally used for a personal project and that became mainstream. This adequately explains the language's *unique* inconsistencies in structure, syntax, and convention, as well as some other language "features". First, it creates a BigInteger of the size 10^{100} by creating a string with the number and inputting this to the BigInteger constructor. Note that the PHP operator for concatenating strings is "." and not "+" like most languages. Then, it iterates over each year, sleeping for a year.

```
function waitForHeatDeath() {
    include('Math/BigInteger.php');

    $years_str = "1";
    for($i = 0; $i < 100; $i++) {
        $years_str .= "0";
    }

    $years = new Math_BigInteger($years_str);
    $big_1 = new Math_BigInteger(1);

    for($i = new Math_BigInteger(0); $i < $years; $i += $big_1) {
        sleep(31536000);
    }
}
```

3.21 Verilog

Verilog is a hardware description language. It can be used in designing and verifying circuits at the register-level of abstraction. By using a 32 bit register containing a value of 10^9, we can loop through this value exponentially 11 times, which will get us an integer value of 10^{99}. And then, we can loop through a separate additional 10 times. This will get us a total integer value of 10^{100}. By using the using `timescale 1s / 1s` command, the time passes once every second in the program. To count once per year, 31536000 seconds need to pass using the `#31536000` command.

```
‘timescale 1s / 1s
module HeatDeathUniverseCounter;
    reg [31:0] W = 32’h3B9ACA00; //10^9
    integer i, j, a, b, c, d, e, f, g, h, k;
    integer val_c = 0;
    always@(W)
    begin
        //10^99
        for(i=0;i<W;i=i+1) begin
            for(j=0;j<W;j=j+1) begin
                for(a=0;a<W;a=a+1) begin
                    for(b=0;b<W;b=b+1) begin
                        for(c=0;c<W;c=c+1) begin
                            for(d=0;d<W;d=d+1) begin
                                for(e=0;e<W;e=e+1) begin
                                    for(f=0;f<W;f=f+1) begin
                                        for(g=0;g<W;g=g+1) begin
                                            for(h=0;h<W;h=h+1) begin
                                                for(k=0;k<W;k=k+1) begin
                                                    #31536000 val_c += 1;
                                                    $display(val_c);
                                                end
                                            end
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end

        //10
        for(i=0;i<10;i=i+1) begin
            #31536000 val_c += 1;
            $display(val_c);
        end
    end
endmodule
```

3.22 R

R is a language specialized in statistical computing and graphics. Surprisingly, R has similar structure as that of Python for loops, which results in the code stub below looking almost identical to that of the Python code stub except a few differences: exponentiation is done with a karat symbol, and replacing *time* (Python) with the conveniently built-in *Sys* (R).

```
wait_for_heat_death_of_the_universe <- function() {
    # Loop through 10^100 years
    for (count in 0:10^100) {
        # Wait for one year (in seconds)
        Sys.sleep(31536000)
    }
}
```

3.23 Swift

Swift is Apple's language for app development, and as such is an easy target for stalling forever if one ever wanted to make an app that does nothing for all time. Since Swift has a limited integer size, this must be implemented with several for loops. For abstraction, this is done with a recursive function.

```
import Foundation
func waitForNYears(_ base: Int, toThe exp: Int) {
  if exp == 0 {
    // Sleep one year
    sleep(31536000)
  } else {
    for _ in 1...base {
      waitForNYears(base, toThe: exp-1)
    }
  }
}
// Wait for (10^10)^10 years
waitForNYears(Int(pow(10.0, 10.0)), toThe: 10)
```

3.24 Dyalog APL

Quite possibly one of the best examples of a "write-only" programming language, APL (and the modern flavor required here, Dyalog APL) uses a non-ASCII character set to allow for incredibly powerful, terse, and illegible matrix and vector operations. While APL has support for staggeringly large arrays and numbers (able to directly represent 10^{100}), the delay function does not have support for such large arguments. A "pedantic" way of working around this would be to compound delay calls through an array and the use of the handy ⍣ operator, delaying at each step. However, we hit an array size limit at 2^{64}, much smaller than the 10^{100} needed here. As such, we must commit what is paramount to heresy in such a beautiful language (with a global-state iteration being wholly unacceptable), and rely on traditional recursive methods. A simple function is presented here, to recursively decrement the right argument of a monadic function, delaying one second at each iteration. While the author would prefer to use the elegance of tacit functions, this is impossible due to the inability to rely on the ⍣ operator for numbers of such size.

```
   wait_for_heat_death ← {{ω=0:⋄∇ω-1⊣⎕DL⊢1} 31536000×1E100}
   wait_for_heat_death ρ0
```

3.25 COBOL

COBOL, misgivingly, does not support sleeping nor does it support a traditional loop structure. To get around this issue, we call the standard UNIX sleep command with an argument of `"infinity"`. By default, sleeping infinitely on UNIX will result in a delay of 9223372036854775807 seconds (about 10^{11} years). To achieve heat death, we must do this repeatedly (roughly 10^{89} times), or by running 8 recursive loops of 10^{10} followed by one loop of 10^{9}.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HEAT-DEATH.
DATA DIVISION.
WORKING-STORAGE SECTION.
    01 my-var PIC X(6) VALUE "Hello!".
    01 cmd-line   pic x(15)  value "sleep infinity" & x"00".
PROCEDURE DIVISION.
MAIN-PROCEDURE.
    PERFORM Child1 10000000000 TIMES
    STOP RUN.
Child1.
    PERFORM Child2 10000000000 TIMES.
Child2.
    PERFORM Child3 10000000000 TIMES.
Child3.
    PERFORM Child4 10000000000 TIMES.
Child4.
    PERFORM Child5 10000000000 TIMES.
Child5.
    PERFORM Child6 10000000000 TIMES.
Child6.
    PERFORM Child7 10000000000 TIMES.
Child7.
    PERFORM Child8 10000000000 TIMES.
Child8.
    PERFORM Child9 1000000000 TIMES.
Child19.
    call "SYSTEM" using cmd-line
    DISPLAY my-var.
END PROGRAM HEAT-DEATH.
```

3.26 Mathematica

Mathematica has a max array size of $2^{31} - 1$, so our wait can not be performed in one operation. Like other approaches, we can split this operation into smaller chunks.

```
Do[
    Do[
        Do[
            Do[
                Do[
                    Do[
                        Do[
                            Do[
                                Do[
                                    Do[
                                        Pause[31536000],
                                    {n, 10000000000}],
                                {n, 10000000000}],
                            {n, 10000000000}],
                        {n, 10000000000}],
                    {n, 10000000000}],
                {n, 10000000000}],
            {n, 10000000000}],
        {n, 10000000000}],
    {n, 10000000000}],
{n, 10000000000}]
```

3.27 Perl

Perl is a general-purpose programming language currently used for a including system administration, web development, network programming, GUI development, and more. As with other languages, PERL comes with a limit on data sizes depending on the version of Perl. For the 32-bit version, the maximum array size is 9007199254740992. So, for the ease of computation, I used a similar approach to previously discussed languages in using 10 nested for loops that each count to 1e10.

```
sub(wait_for_heat_death_of_universe){
    for ( 0;  10000000000; 1 ) {
        for ( 0;  10000000000; 1 ) {
            for ( 0;  10000000000; 1 ) {
                for ( 0;  10000000000; 1 ) {
                    for ( 0;  10000000000; 1 ) {
                        for ( 0;  10000000000; 1 ) {
                            for ( 0;  10000000000; 1 ) {
                                for ( 0;  10000000000; 1 ) {
                                    for ( 0;  10000000000; 1 ) {
                                        for ( 0;  10000000000; 1 ) {
                                            sleep(rand(31536000))
    }}}}}}}}}}
}
```

3.28 Lua

Lua is a scripting language built on top of C. Lua does not support a sleep function by default, so we start by defining one. Note that because these *os* calls are expensive, a more optimized solution would use and call a C function to block instead of busy waiting as shown here. However, the spirit of this exercise requires sticking to Lua, so we write the following function:

```
function wait(time)
    local duration = os.time() + time
    while os.time() < duration do end
end
```

The maximum value Lua supports is a 64-bit integer, meaning that we need to use a set of nested loops to wait 10^{100} years. One may choose to indent for stylistic reasons, but we show the loops collapsed here for ergonomics when reading on a page.

```
for count1 = 0, 10000000000 do
for count2 = 0, 10000000000 do
for count3 = 0, 10000000000 do
for count4 = 0, 10000000000 do
for count5 = 0, 10000000000 do
for count6 = 0, 10000000000 do
for count7 = 0, 10000000000 do
for count8 = 0, 10000000000 do
for count9 = 0, 10000000000 do
for count10 = 0, 10000000000 do
    wait(31536000)
end
end
end
end
end
end
end
end
end
end
```

3.29 Julia

In this case, the structure is similar to the Python 3 approach - create a range object to delimit the limits of heat death, nicely assign Julia to sleep for a year (in seconds) each iteration, and let the program go to task. For no other reason than aesthetics, the implementation below uses one-line coding.

```
for count in range(1, 1e100); sleep(31536000); end
```

3.30 Rust - Looping

Rust is a statically typed language pioneered by Grayden Hoare in 2007 and then incubated by Mozilla for use in the Mozilla Firefox web browser. Today, Rust empowers developers to write systems programs that emphasize memory safety and performance. Rust does not natively support arbitrary precision integers: these datatypes are provided by third-party libraries (crates) in the ‘crates.io‘ ecosystem. While Rust does not provide arbitrarily sized integers in the standard library, it does provide unsigned integers up to 128 bits in size, making a nested loop solution more succinct than other languages that only support up to 32 bit or 64 bit integers.

```
fn main() {
    for _ in 0..10_u128.pow(33) {
        for _ in 0..10_u128.pow(33) {
            for _ in 0..10_u128.pow(33) {
                std::thread::sleep(std::time::Duration::from_secs(60 * 60 * 24 * 365 * 10));
            }
        }
    }
}
```

3.31 Rust - Arbitrary Precision

While looping might be more succinct for this particular implementation, it is not as flexible as an arbitrary precision approach. To imitate an arbitrary precision without installing a third-party library, we can use an array of unsigned 128bit integers to achieve the required 333 digits of precision. This approach will simply increment our packed integer array until the target value is reached, sleeping one year between increments.

```
fn main() {
    let mut count: [u128; 3] = [0, 0, 0];
    let target: [u128; 3] = [u128::MAX, u128::MAX, 2u128.pow(77)];

    while count != target {
        for i in count.iter_mut() {
            *i = i.wrapping_add(1);

            if *i != 0 {
                break;
            }

            std::thread::sleep(std::time::Duration::from_secs(60 * 60 * 24 * 365));
        }
    }
}
```

3.32 MIPS Assembly

MIPS (Microprocessor without Interlocked Pipeline Stages) assembly is a RISC (Reduced Instruction Set Computer) ISA (Instruction Set Architecture). It is commonly used in the undergraduate computer engineering teaching curriculum. This approach is very similar to the Java approach. Since the MIPS registers are 32-bit, a maximum number of 2,147,483,647 can be stored. For ease, we use 11 nested loops counting till 10^9 with a one year waiting period for each iteration.

```
.macro sleepSecond
    # wait for 1 second
    li $a0, 1000
    li $v0, 32
    syscall
.end_macro

.macro Terminate
    # end the program
    li $v0, 10
    syscall
.end_macro

li $t0, 0

# wait for one year
waitYear:
    addi $t0, $t0, 1
    sleepSecond
    beq $t0, 31536000, looper1
    j waitYear

# initialize all of the counting variables
li $t1, 0
li $t2, 0
li $t3, 0
li $t4, 0
li $t5, 0
li $t6, 0
li $t7, 0
li $t8, 0
li $t9, 0
li $s0, 0
li $s1, 0

looper1:
    addi $t1, $t1, 1
     beq $t1, 1000000000, looper2
     li $t0, 0
     j waitYear
```

```
looper2:
     addi $t2, $t2, 1
     beq $t2, 1000000000, looper3
     li $t1, 0
     j looper1

looper3:
     addi $t3, $t3, 1
     beq $t3, 1000000000, looper4
     li $t2, 0
     j looper2

looper4:
     addi $t4, $t4, 1
     beq $t4, 1000000000, looper5
     li $t3, 0
     j looper3

looper5:
     addi $t5, $t5, 1
     beq $t5, 1000000000, looper6
     li $t4, 0
     j looper4

looper6:
     addi $t6, $t6, 1
     beq $t6, 1000000000, looper7
     li $t5, 0
     j looper5

looper7:
     addi $t7, $t7, 1
     beq $t7, 1000000000, looper8
     li $t6, 0
     j looper6

looper8:
     addi $t8, $t8, 1
     beq $t8, 1000000000, looper9
     li $t7, 0
     j looper7

looper9:
     addi $t9, $t9, 1
     beq $t9, 1000000000, looper10
     li $t8, 0
     j looper8
```

```
looper10:
     addi $s0, $s0, 1
     beq $s0, 1000000000, looper11
     li $t9, 0
     j looper9

looper11:
     addi $s1, $s1, 1
     beq $s1, 1000000000, exit
     li $s0, 0
     j looper10

exit:
     Terminate
```

3.33 NetFuck

Unfortunately, as the base *brainfuck* language does not provide the ability to 'sleep' for a certain amount of time, we need to look towards an extension of the language that does, such as *Alarm Clock Radio* or the language we have selected, *NetFuck*. Because of the limited instruction set (including no direct *if* operators, etc.), and because we assume some typical limitations to *brainfuck*-derived machines (32-bytes per memory cell, $\tilde{1}00$ memory cells), we need to rely on some pretty brute-force methodology: iteratively adding value to memory cells until we have 10^9 stored in 11 cells; do the same for our other large number counters; use those set values as counters for our loops and iterate through, sleeping for 10ms on each iteration. It amounts to an astonishing amount of un-optimized computation, meaning that on a slow single-cycle CPU the computation alone would be enough to reach the heat death of the universe. With the addition of a timer, we can ensure heat death will happen faster than the end of computation even on high-powered machines. On the bright side, it is amazingly memory efficient - we only require 15 32-byte registers.

```
Set 10^9 in cell 12:
>>>++++++++++
[
 >++++++++++
 [
  >++++++++++
  [
   >++++++++++
   [
    >++++++++++
    [
     >++++++++++
     [
      >++++++++++
      [
       >++++++++++
       [
        >++++++++++
        <-
       ]
       <-
      ]
      <-
     ]
     <-
    ]
    <-
   ]
   <-
  ]
  <-
 ]
 <-
]

Pre-load 10^9 into the preceding 11 cells, zero cell 12
>>>>>>>>
[
 <+<+<+<+<+<+<+<+<+<+<+
 >>>>>>>>>>>-
]

Set 31536000 in the cell 15:
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
+
[
 >++++++++++++++++++++++++
 [
  >++++++++++++++++++++++++
  ++++++++++++++++++++++++++
  ++++++++++
  [
   >+++++++++++++++++++++++
   +++++++++++++++++++++++++
   ++++++++++++
   <-
  ]
  <-
 ]
 <-
]

Move 31536000 back into cell 12, zero cell 15:
>>>
[
 <<<+
 >>>-
]

Store 100 in cell 13, move to start:
<++++++++++
[
 <++++++++++
 >-
]
<<<<<<<<<<<<<

Use 10 consecutive timers, special to NetFuck, of 10 ms * [cell 13] = 1000 ms each.
Loop 3153600 times, then 10^9 times, 11 times.
[
 >
 [
  >
  [
   >
   [
    >
    [
     >
     [
      >
      [
       >
       [
        >
        [
         >
         [
          >
          [
           >
           [
            >~~~~~~~~~~
            <-
           ]
           <-
          ]
          <-
         ]
         <-
        ]
        <-
       ]
       <-
      ]
      <-
     ]
     <-
    ]
    <-
   ]
   <-
  ]
  <-
 ]
 <-
]
```

For the sake of conciseness, here is the same NF code as above, but without white-spacing (line-wrapping is done for ease of reading).

```
>>>++++++++++[>++++++++++[>++++++++++[>++++++++++[>++++++++++[>++++++++++[>+++++
+++++[>++++++++++[>++++++++++<-]<-]<-]<-]<-]<-]<-]<-]>>>>>>>>[<+<+<+<+<+<+<+<+<+
<+<+>>>>>>>>>>>-]+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++[>++++++++++++++++
++++++++[>++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++[>++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++<-]<-]<-]>>>[<<<+>>>-]<+++++
+++++[<++++++++++>-]<<<<<<<<<<<<<[>[>[>[>[>[>[>[>[>[>[>[>~~~~~~~~~~<-]<-]<-]<-]<
-]<-]<-]<-]<-]<-]<-]<-]
```

3.34 brainfuck

We just teased at how to make *brainfuck* work without relying on an infinite loop - under very specific assumptions. If we restrict operation to a single-cycle CPU without branch-prediction and other compiler optimizations, running at low frequencies (1MHz), we can mimic sleeping for 1 second by simply running one million computations each loop and forcing the processor to be delayed for one second per iteration.

3.35 Befunge-98

Befunge is a programming language in which an instruction pointer moves along a two-dimensional grid. Using Befunge-98's execute extensions we can call the Unix command "sleep 31536000" to sleep for a year. The row of "!" characters in the source code act as counters; as execution progresses they are modified to store the current iteration number. The range of printable ASCII characters limits the maximum value for each individual counter, but by chaining them and adding carrying logic we can build a timer that counts to 90^{53} years, a value which exceeds the desired wait duration.

```
>"sleep 31536000"=04g1+:13g`!v
^                        <p40_$b3*03p>23g03gb3*-4p03gb3*-1+:33g`#@_b3*+03pv
                                     ^                 _v#`g31:+1g4-*3bg30<
I}!5                     ^                     p4-*3bg30<
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

3.36 Minecraft Redstone

Minecraft is a block-based creative building game with a Turing complete circuit-building system using Redstone and Redstone components. The designed circuit is made up of a series of chained Modified Etho Clocks with Hoppers (MECHs) (See figures 2 3). Each of these MECHs consists of an unmodified Hopper Clock with an added circuit that allows it to be chained when one's observer is pointed at another's final repeater.

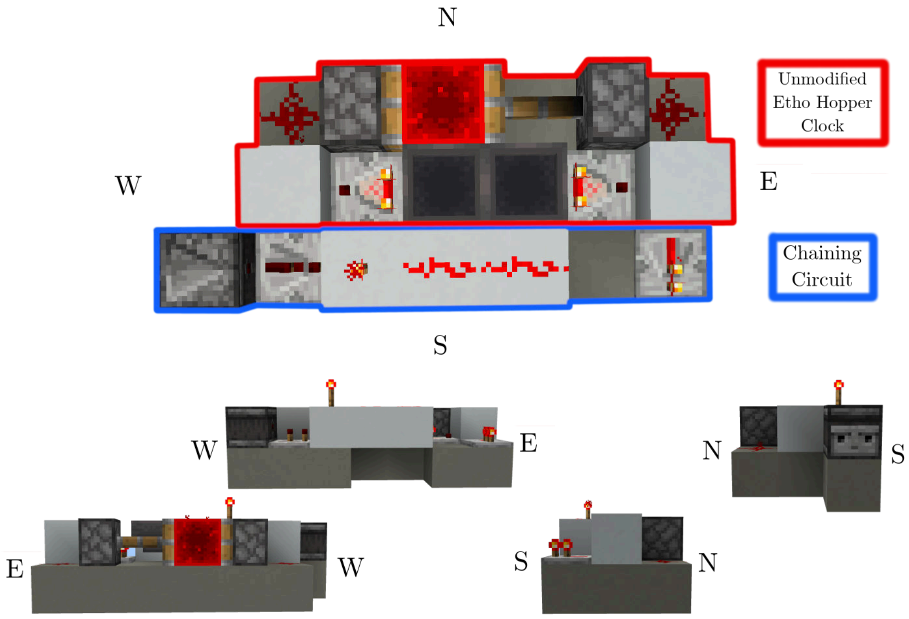


Figure 2: An in-game Modified Etho Clock with Hoppers from several angles, annotations are added for visual comprehension.

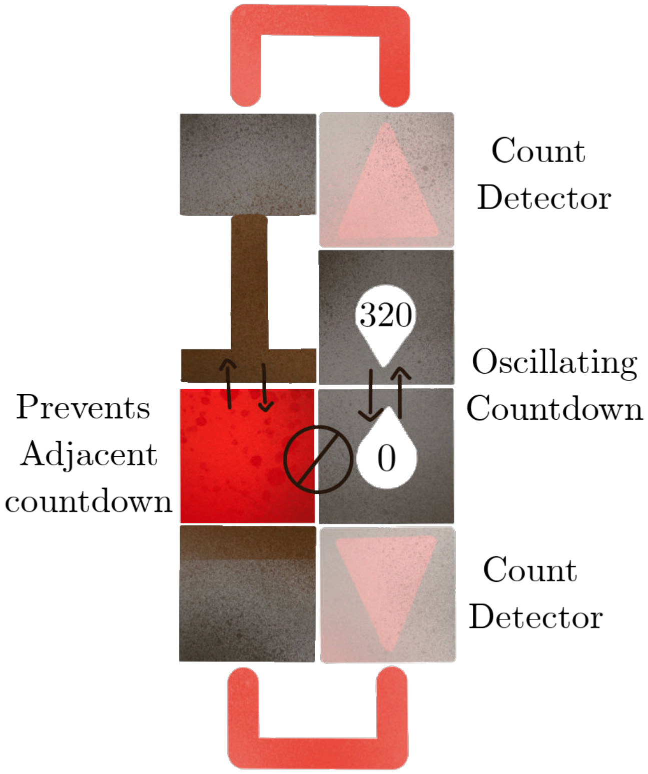


Figure 3: A diagram for an unmodified Etho Hopper Clock with labels added.

The first module in the series is an unmodified Etho Hopper Clock. Hoppers in Minecraft move items once every 8 in game ticks and a hopper can hold up to five 64 item stacks. The output from this clock triggers once every two flip-flops as the items must travel from one hopper to the other and back to the original hopper. The conversion math is shown below.

$$\frac{5\text{ stacks}}{1\text{ flipFlop}} \times \frac{64\text{ items}}{1\text{ stack}} \times \frac{8\text{ ticks}}{1\text{ item}} \times \frac{1\text{ second}}{20\text{ ticks}} \times \frac{2\text{ flipFlops}}{1\text{ cycle}} = 256\frac{\text{seconds}}{\text{cycle}}$$

Each MECH is designed to be chainable thus the time between pulses multiplies exponentially as more are tacked on. Conceptually, each can be thought of as a gearbox with a ratio of the number of items multiplied by two as each item travels into and out of the second hopper every cycle. Each MECH is filled full with five 64 item stacks for a total gear ratio of 640:1. This means for every 640 pulses the clock takes in, it will output one pulse. Effectively, each MECH multiplies the total clock's time by the number of items it holds multiplied by two. From this we find that

$$\text{first cycle time} \times (2 \times \text{num items})^{\text{num MECHs}} = \text{total time}$$

We can then plug in known values and solve for the number of MECHs:

$$\text{num MECHs} = \log_{640}\left(\frac{10^{100} \times 3.1536 \times 10^{7}}{256}\right) = 37.4498065\text{ MECHs}$$

Thus, we need 38 MECHs and one Unmodified Etho Hopper Clock to reach the heat death of the universe.

Minecraft is a game that needs to run on a standard computer so it has limits. Surely that must keep it from being able to count to the end of the universe. The biggest problem is that, redstone only works within 21 chunks or 336 blocks of the player on a standard world. Fortunately, each MECH is only 7 blocks long meaning this circuit would be only 272 (adding 6 for the unmodified hopper clock) blocks long, fitting comfortably within that distance without modification.

Climate Science

22 Towards Cloud Computing

Alex Xie and Alan Hsu

Keywords: cloud computing, deep learning, machine learning, computer vision, toddlers, kahoot, clout, drip, label smoothing, data science, animals, atmospheric sciences, meteorology

23 Ecological Memory Management: Beyond Garbage Collection

Erik Derohanian, Dann Toliver and Saul Field

Keywords: Garbage Collection, Compost, Memory Management, Organic Computing

24 Infrastructure-as-PowerPoint: A No-Code Approach to Cloud Infrastructure Management

Tobias Pfandzelter

Keywords: cloud computing, powerpoint, slideshow bob

25 On the Possibilities and Challenges of Organic UAV-Assisted MEC

S. Wallow, Cardi Nalle, Robin, P. Cock, and Sky Lark

Keywords:chick-a-dee, caw caw, chirp chirp, hoot hoot

Towards Cloud Computing

Alex Xie
Candyland Daycare
Edison, NJ 08820
gagagoogoo@candyland.cs.edu

Alan Hsu
Apple Montessori
Edison, NJ 08820
googoogaga@montessori.cs.edu

Abstract

Every day, millions of our fellow young young adults spend countless toddler-hours struggling to identify objects from the clouds in the sky, losing out on their childhood years. To this end, we introduce DRIP *for* CLOUT*, a novel benchmark for automated cloud recognition, and we utilize deep learning models to achieve state of the art results on this task. This marks a definitive step towards alleviating the stress of cloud-gazing on toddlers, allowing them to focus on other aspects of life, such as bed-wetting, learning their ABCs, and writing SIGBOVIK papers.*

1. Introduction

In the past decade, there has been a great deal of interest in the adult community in cloud computing [5][6]. This paper is not about that.

Within the toddler community, there has recently been a great deal of interest in cloud computing. During daycare, we are often given a set of toys that over time are increasingly boring and decreasingly sanitary. Thus, we resort to satiate our interests by looking out the windows, longing to explore the wilderness. The closest proxy to such a desire are the ever-changing clouds that loom over the skies, and something that we can stare at all day.

We all know that one of the long standing issues of being a toddler is the inability of identifying clouds. However, in the advent of modern deep learning architectures, we are in a better position to address this complex problem.

In our work, we first introduce the **CLOUT** task (**CL**oud **O**rganization **U**sing **T**oddlers). We then introduce the **Kahoot** data collection paradigm and the **DRIP** (**D**ataset for **R**ecognit**I**on of Cloud **P**atterns), a novel dataset for the **CLOUT** task. Next, we propose several baseline models for the **DRIP** dataset, and provide post-train evaluation metrics on these models. Ultimately, we find that previous researchers could not handle CLOUT because they did not have our DRIP.

2. Related Work

Many toddlers have already made significant research advancements in this field [1]. For example, in [8] researchers have designed an efficient algorithm 1 for identifying clouds, under the supervision of annoying experts, who often refer to themselves as adults:

Algorithm 1 Toddler Forcing

```
Require: Expert E[a], Toddler T[b], observed cloud C
Ensure: Description D
  E encourages T to describe what they see in C
  Init D with what T thinks it is.
  while D does not have a good description do
      E and T take turns describing the cloud.
  end while
```

[a] The Expert is parameterized by an underfitted 100 billion neuron network that maps concepts to semi-grammatically incorrect sentences

[b] The Toddler is parameterized by an untrained 100 billion neuron network that maps concepts to gurgling noises and occasionally grammatically incorrect sentences

Ideally, we would collect our data directly from toddlers, as described in Algorithm 1. However, this is an NP-hard task.[1] In the last year, we have made a major breakthrough with state-of-the-art algorithms, utilizing sweet reinforcement learning (sweet RL) techniques to train toddlers to behave as we want. Using candy such as *Sour Patch Kids* or *Hershey's Chocolate* as the reward, and simulating discounted reward functions by slowly eating the candy[2], we can incentivize the toddler to speak.

Nonetheless, even with these tasty algorithms training toddlers to identify clouds is unreasonably slow and, quite frankly, impractical and inhumane. Thus, we will relinquish such an ambitious goal and resort to using computers, which are much more obedient.

[1] Have you ever tried getting a toddler to do what you actually wanted them to do?

[2] Effectively stealing candy from a baby

	Kahoot	Socratic Seminar	Raising Hands	Owl	4chan
Virtual	✓	✗	✗	✗	✓
Latency (ms)	10	1000	5000	600000000	10000
Used in education	✓	✓	✓	✓	✓
Engaging Music	✓	✗	✗	✗	✗
Contains "K" in name	✓	✗	✗	✗	✗
Custom Username	✓	✗	✗	✓	✓
Used by Wizards	✗	✗	✗	✓	✗
Non-GMO	✗	✗	✗	✓	✗
Shames failure	✓	✗	✗	✗	✓
Onii-chan	✗	✗	✗	✗	✓
Hotel?	Trivago	Trivago	Trivago	Trivago	Trivago

Table 1: Comparison of Data Collection Paradigms. Our Kahoot paradigm is most robust, followed by Owl.

3. The CLOUT Task

We now introduce the main objective of this paper, the CLOUT problem: *Given a labeled dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, train a deep network classifier to best approximate the target function $f : X \rightarrow y$, mapping cloud instances to their labels.*

4. The DRIP Dataset

As directly sampling from toddlers is computationally infeasible (and possibly illegal?), we approximate the mind of a toddler by collecting data from undergraduate computer science students, which has repeatedly been shown to be an asymptotically tight approximation [3]. Following this method, we acquire DRIP.

4.1. Cloud Image Source

We obtain our images from the Singapore Whole Sky IMaging SEGmentation Database (SWIMSEG), a dataset of 1013 images originally labeled for cloud segmentation [4].

Below are 4 sample images, one from each of the 4 classes:

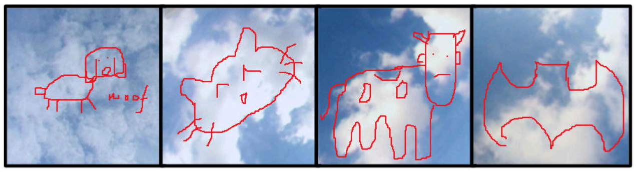


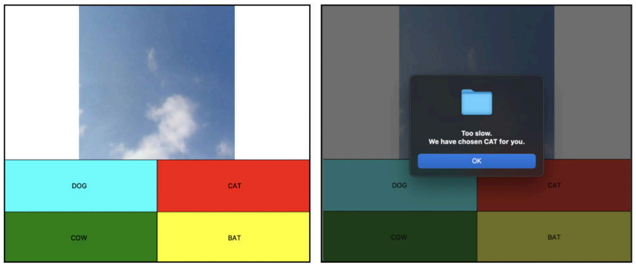
Figure 1: Sample Images from SWIMSEG, with labels (from left to right) dog, cat, cow, and bat

4.2. Kahoot Data Collection Paradigm

As data collection was conducted during the pandemic, we needed to obey the social distancing L_∞ metric. As such, we designed the Kahoot Data Collection Paradigm, which allowed our undergraduate subjects to participate remotely. The students helped us label images with 4 classes, namely *bat*, *cat*, *cow*, and *dog*. Additionally, students are familiar with this platform, and also have the flexible choice to anonymize their names if they so choose.

Further, to appease undergraduate students seeking a career in research, we offered as an additional incentive to credit them as coauthors on this paper. However, we never *pinky-promised*, and as such their names remain conspicuously absent.

To demonstrate the effectiveness of our choice, we display the comparison table of other data collection paradigms in table 1.



Too slow.
We have chosen CAT for you.

Figure 2: **Left:** Kahoot data collection setup. **Right:** After ten seconds, Kahoot times out and selects an answer for the user.

4.3. Data Quality

We leave an exploration of data quality to future work. We assure the reader that data quality is by far one of the goals of this work.

[3] For example, `https : / / www . instagram . com / cmudaddythicc/?hl=en`

4.4. Comparison to Existing Datasets

We present in Table 2 a comparison between DRIP and related cloud computing datasets. Unfortunately prior to this work, we did not know adults also used the term "cloud computing." As such we were fooled into running preliminary experiments on Google and Microsoft Azure's knockoff datasets for what they consider to be "cloud computing."

	DRIP	IN	GC	AP
Has clouds	✓	✓	✗	✗
Used for cloud computing	✓	✗	✓	✓
Introduced by this paper	✓	✗	✗	✗
Random labels	✓	✗	✗	✗

Table 2: DrippingCap, a comparison of DRIP against related cloud computing datasets, ImageNet (IN), Google Cluster Dataset (GC), and Azure Public Dataset (AP)

5. Models

A great deal of recent work in machine learning and adjacent fields has focused on eliminating biases from datasets and models. Specifically, there has been much work towards preventing models from learning harmful biases against certain groups. In this paper, we extend this by preventing our model from learning anything at all. In an effort to accomplish this, we propose *extreme* label smoothing (XLS).

Label smoothing [7] modifies the label distribution by interpolating with some $\epsilon \in [0, 1)$ between the true one-hot label distribution $q(k \mid x)$ and a uniform prior $u(k)$:

$$q'(k \mid x) = (1 - \epsilon)q(k \mid x) + \epsilon u(k) \qquad (1)$$

This naive (and frankly discriminatory approach) leaves much room for the model to learn undesirable associations from the data - for example, that cows are fatter than dogs. Hence, in extreme label smoothing, we propose to set $\epsilon = 1$ in Equation (1). Where label smoothing penalizes overconfidence, extreme label smoothing destroys the model's self-confidence altogether. Machine learning can't bully us if we bully it first.

Ultimately, rather than training a discriminator, we wish to train a unifier that brings all classes together in Marxist harmony [2].

6. Experimental Evaluation

Our accuracy is very good, trivial by Jensen's [4].

[4] Source: `https://www.youtube.com/watch?v=-fGKrYq8_dk`

7. Results and Discussion

Surprisingly, the confusion matrix for all of our experiments were the same, shown in figure 3.

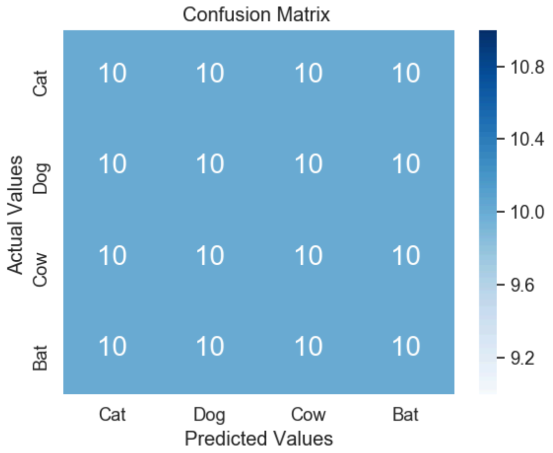


Figure 3: Confusion Matrix of our experiments

This can be attributed to the fact that our research assistants were toddlers who only had ten fingers with which to record results. We hope that in the future, we can get more fine-grained results by having our researchers use their toes as well.

8. Conclusion

We conclude that our state-of-the-art machinery is able to predict the shape of clouds extremely accurately, thus we toddlers are bing chilling [3].

9. Future Work

We have solved the millennium problem of identifying clouds. We expect toddler researchers to be very happy because they can now move onto other research challenges, such as estimating the rate of grass growth and the rate of bed-wetting.

10. Acknowledgements

We would like to thank the Vision and Interaction Group at the National University of Singapore for generously (and quickly) providing us with the SWIMSEG dataset. We would also like to thank our high school physics teacher for inspiring us to use the Kahoot data collection paradigm. Finally, we would like to thank (but unfortunately for them, are not going to) the undergraduate students that helped us label the SWIMSEG dataset.

References

[1] Experienced childcare — edison, nj — candyland academy. `https://www.candylandacademy.com/`, 2005.

[2] Samuel Albanie, Sébastien Ehrhardt, and João F. Henriques. Stopping gan violence: Generative unadversarial networks. In *SIGBOVIK 2017*.

[3] John Cena. Eating ice-cream — bing chilling. `https://www.youtube.com/watch?v=AWOyEIuVzzQ`, 2021. [Online; accessed 1-April-2022].

[4] Soumyabrata Dev, Yee Hui Lee, and Stefan Winkler. Color-based segmentation of sky/cloud images from ground-based cameras. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(1):231–242, 2017.

[5] George Favaloro. 1996 compaq business plan. , 1996. [Online; accessed 1-April-1996].

[6] Google. why is cloud computing important - google search. `https://www.google.com/search?q=why+is+cloud+computing+important&source=hp&ei=rX0-YpOYDbSF9PwPov26qAk&iflsig=AHkkrS4AAAAAYj6LvQWkMZjUBhAH-3TBErYa4HQntN3k&oq=why+is+cloudj+computi&gs_lcp=Cgdnd3Mtd2l6EAMYADIECAAQDTIECAAQDTIECAAQDTIECAAQDTIECAAQDTIECAAQDTIECAAQDTIECAAQDTIECAAQDTIECAAQDToLCA` `sclient=gws-wiz`, 2022. [Online; accessed 25-March-2022].

[7] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[8] Penny Whitehouse. Finding animal cloud shapes with kids — mother natured. `https://mothernatured.com/animal-play/finding-animals-in-clouds/`, 2013. [Online; accessed 1-April-2013].

forgot to remove page numbers
ach

Ecological Memory Management: Beyond Garbage Collection

April 1, 2022

Erik Derohanian[a,c,d,o,‡,*,@] Saul Field[c,o,†,‡,*,#,@] Dann Toliver[b,c,d,o,‡,*,@]

[a]Institute of Institutional Institution
[b]Workplace Corporation
[c]Computer Science Cabal
[d]College of Collegiate Collages
[o]Organic Computing Association
[†]Invisible College of the Rosy Cross
[‡]Authors contributed equally to this work
[*]Authors deny any connection to this work
[#]Author's actual name, no joke
[@]Correspondence: dev.null@example.com

Abstract: There's too much garbage in the world already – we shouldn't add more to it. We propose a new system for memory management that reuses and recycles whatever it can, and composts the remainder. The recycling centre salvages objects that would otherwise end up in the landfill of `/dev/null`, providing automated object pooling. And unlike compacting garbage collectors that merely squish things, our composting garbage collector actually converts garbage into entropy. We examine a variety of techniques for entropy generation within the compost heap. We explore practical implementations of our composting collector on current hardware, and point toward the possibilities afforded by future hardware designs. Finally, we show that with appropriate application of reuse, recycling, and composting, we can completely eliminate unwanted digital waste.

Introduction to Ecological Memory Management

A Manifesto

Ecological Memory Management is a new field of of memory management that takes its environmental responsibility seriously and focuses on reuse, recycling, and composting instead of constantly allocating and disposing of objects. Every day, exabytes of data are irresponsibly garbage collected into bit buckets where they will never be used again[1].

Ecological Memory Management is an aspect of organic computing, which encourages the use of local resources to solve problems whenever possible. Entropy is a common resource request, for instance, and many processes pester the operating system quite frequently with random calls. We show that with a little work by our composting garbage collector, the garbage created by the typical process provides a high quality source of entropy that should be more than sufficient for its needs[2].

We also show that the work put into constructing values and data structures can be saved through reuse and recycling. This provides a number of benefits, such as automatic object pooling, and minimizes wasteful bit flips.

[1] Those bits could be the works of artists, the deleted tweets of politicians, or the carefully crafted structures assembled by programmers, and it is our duty as denizens of our data centers and as stewards of our servers to ensure that those stale bits are disposed of responsibly, with the care and respect that they deserve.

[2] When we run our processes in completely sterile environments, devoid of rich digital detritus, is it any wonder that they crash when the slightest thing goes wrong?

Ecological Memory Management asks computer users and their processes to strive to follow the Boy Scout Rule: leave the computer better than you found it, exploit as few system resources as possible, and give bits and bytes back whenever possible to be shared, reused, recycled, or composted. By working together with our co-located neighbors, we can ensure that our server farms can continue to produce high quality digital comestibles far into the future.

The Memory Landscape

Organic computing starts on the farm. The memory managed within a computational process — the land on its farm — cycles between four states throughout the lifecycle of the process, as seen in Figure 1.

- **Fresh Soil**. Fresh memory available to make new allocations. Will become `planted soil` when an allocation is made.
- **Planted Soil**. Memory occupied by planted objects. This is the working heap for the process and its threads. When the compost heap expands, all live objects are transplanted to a new patch of `fresh soil`, and the old patch of `planted soil` is plonked onto the end of the `compost heap` and begins breaking down[3].
- **Compost**. Memory currently being broken down and converted into entropy. Not directly usable, but will eventually become `mulch` once it meets a suitable level of randomness. Will be transformed by various *organisms* which have been *evolved* for this purpose.
- **Mulch**. Memory that has been sufficiently broken down and is available for entropy requiring operations. As bytes are read, the `mulch` is turned into `soil`, and the cycle begins again. `Mulch` can be instantly used as soil if there is an urgent need for new memory.

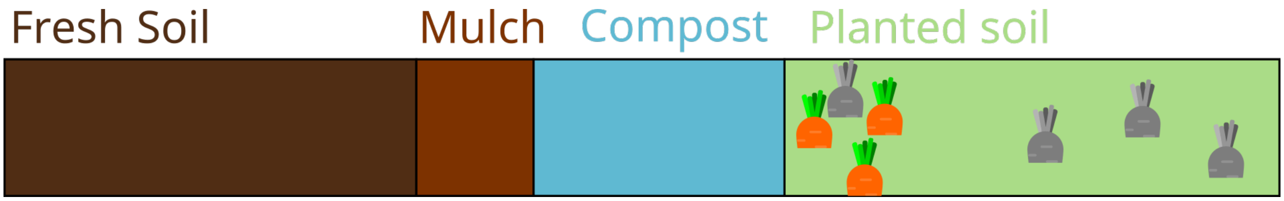


Figure 1: Like a gelatinous cube squidging through long twisty passages, the compost heap lumbers on.

These four states[4] encompass everything within the ecological memory management process.

[3] When you practice organic computing, your process interconnects with others through the rich loamy soil, full of nutritious hummus, and your process is a pita chip or some broccoli and it just came out of the fryer or the farmer's market tote you've been carrying carefree down a summer street wondering where your next stop will be: to the Cheshire cheesery, or maybe to Mabel's, where the dreamy bloke with long freckled arms serves hot buttered rolls and you've thought about it but never gotten up the nerve to ask about his accent, because the rolls are really quite good and it might cause a moment but not necessarily the good kind of moment it could be the bad kind of moment, like the kind where you don't want to go back anymore, and that would be a shame because the rolls really are quite good, and in the end you did stop and have one, and he was there and you didn't ask, and now you are home and you are dipping the broccoli and definitely not the fresh pita chip into the hummus, and it is delicious, and this is exactly what organic computing is like.

[4] Corresponding to the four layers of the ecological lifecycle: growth, death, decay, and spontaneous generation.

When fresh soil or mulch are needed, the compost heap runs. Under normal conditions this provides ample room for process evolution. If additional soil is required for new allocations, the soil patch can be extended by acquiring more land from the county (the operating system).

We can also fall back to the operating system if randomness is required beyond what is provided by our local mulch supply[5].

The Recycling Centre

Our recycling model provides for efficient reuse of both data structures and values, bringing the benefits of object pooling to the runtime level, rather than requiring the programmer to do the work explicitly.

The computational and memory manipulation work of constructing objects is typically lost after that object has been deallocated, so that in addition to those bytes being sent to the landfill the work itself is also wasted.

Instead we can *reuse* these components for new objects[6]. In cases where there is not an exact match, we can *recycle* components and turn them into exactly what we need.

First, we describe what an object actually *is*. In a C-like language, for instance, an object could be as simple as a box:

```
struct box {
    uint8_t type;
    void *value;
};
```

The `type` field represents the underlying type of the value (such as `boolean`, `float`, `tuple`, etc.). The `value` field is a pointer to the block of memory representing the actual object data.

Setting up our objects this way creates a clean separation that allows reusing boxes and values independently from each other. Some may object[7] to having a pointer-sized memory overhead for every object type, including integers. This is a reasonable objection, but it is ultimately a small price to pay for a fully reusable and recyclable object representation.

Some additional memory overhead is required for the recycling centre as well. This will take up a fixed space in memory, and will consist of stacks[8] of boxes and values, categorized by object type (see Fig. 2).

When a dead object is discovered, a pointer to its box is added to the corresponding box stack, and its value to the corresponding value

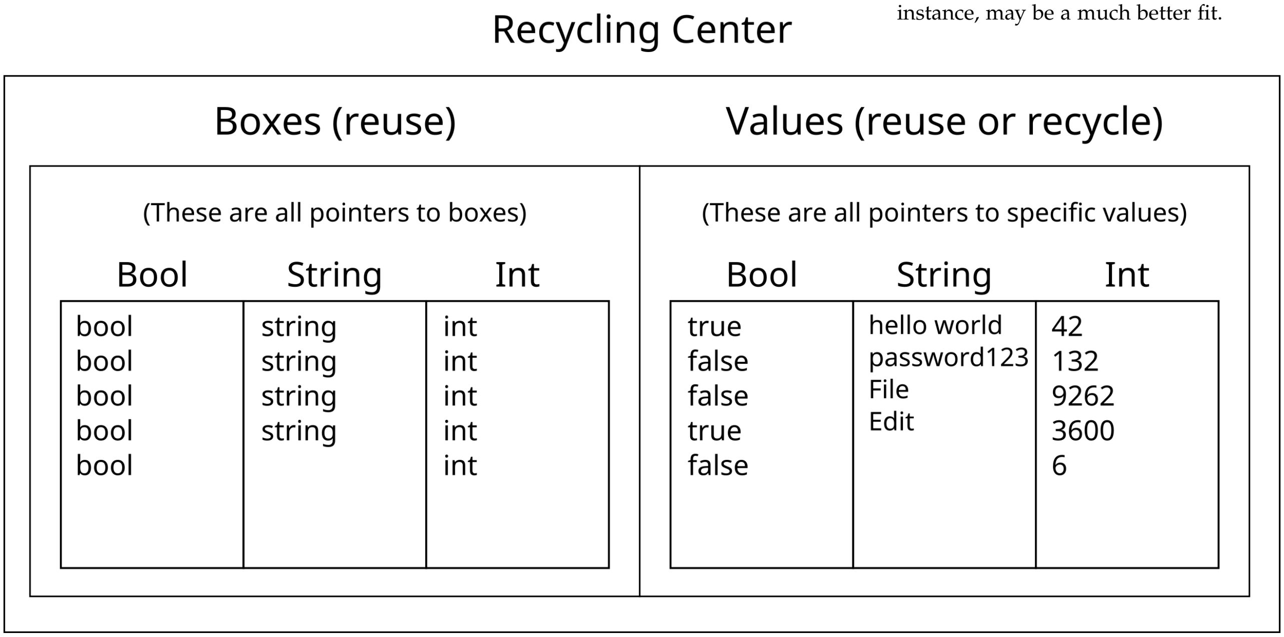
[5] And if blessed with a bountiful mulch harvest, the process can provide excess entropy back to the OS for use in other processes

[6] In practice the first step is to *reduce* usage. Ask yourself before your next allocation: do you really need that object, or could your program get by without it?

[7] Pun very much intended. In fact, this sentence was revised three times specifically to make this work.

[8] Pun not intended. We're talking about warehouses and boxes here, not computer stuff.

stack. If a stack is full, its oldest pointer goes away[9].



Recycling Center

Boxes (reuse)

(These are all pointers to boxes)

Bool	String	Int
bool	string	int
bool	string	int
bool	string	int
bool	string	int
bool		int

Values (reuse or recycle)

(These are all pointers to specific values)

Bool	String	Int
true	hello world	42
false	password123	132
false	File	9262
true	Edit	3600
false		6

Figure 2: The gnomes' recycling centre, an accredited organic computing recycler.

Reuse

When a new object is allocated, the gnomes[10] first check the warehouse for a matching box. If one is found it is removed from the stack and used, otherwise a new one is bought from the store and deposited in fresh soil.

In either case, the pointer to that box is handed to the application and is no longer under management by the gnomes. By applying this recursively to the data structures in our system, object pooling[11] is provided at the language level.

Values work similarly: if a matching value is found, its pointer is removed from the stack and handed to the application. However, there is a lower chance that an identical value exists in the warehouse, so a second layer of processing is provided for values that need to be transformed a bit[12].

Recycle

As we saw, when a new value is requested the appropriate value stack[13] is checked, and if found then that value is tucked into the box. Otherwise, a new value must be bought from the store. But what if we have a bunch of values that are made of the same material as our desired value (i.e. it is of the same type), but isn't quite exactly

[9] As noted previously, these aren't actually stacks. A ring buffer, for instance, may be a much better fit.

[10] These are the workers in the recycling centre. They could also be elves. This is implementation dependent, unless that implementation is GNOME.

[11] Further work is required to determine whether a pool would be a better choice of abstraction than a warehouse.

[12] Or a nibble.

[13] Choice of stack implementation left to implementer. Being organic, we choose trees.

what we need? No need to rush off to the store yet! We can do a bit of processing on a value in the warehouse to recycle it into the requested value. This might sound a bit too involved for something like an integer, but think about larger structures such as strings, lists, and tuples. If we can get the exact value we need by simply stitching in a few bits, this could potentially save us from making a large allocation from scratch.

This approach has some nice benefits. Instead of lugging the entirety of the data from the central silo to where we need it, we take the computation "out into the field". This avoids the von Neumann bottleneck[14], and also makes our recycling problem "embarassingly parallel"[15]. There are a wide array of potential extensions: we could implement this with traditional parallel hardware such as GPUs or multiple cores, or something more radical like GreenArrays[16], or the Movable Feast Machine[17]. Alternatively, we could treat memory as a 2D grid and determine the rules for a cellular automata that will converge to our desired grid state, such as in [18]. This could potentially be supplied by the hardware, much like the "scrubber circuit" in ECC memory.

The Compost Heap

Composting the garbage created by our process means generating something useful from the waste. In particular, we take advantage of *bitrot* to convert that waste into entropy. Entropy is useful as input to a wide variety of processes, including cryptographic operations, machine learning systems, data science, probabilistic programming, and differential privacy applications.

Our composting memory management returns unused memory to its natural state of entropy, allowing it to be consumed as input to entropy-seeking functions and reducing reliance on out-of-process entropy generation methods.

It moves continuously, slurping in objects as it makes its way linearly through memory, transplanting live objects safely into reclaimed land on its far end and composting everything else.

When the compost heap claims dead objects into its fold it releases byproducts, in the form of orphaned child objects[19] and values that are no longer reachable from the roots, and these are captured by the gnomes and stored in their warehouse for recycling and reuse. The compost heap answers the question, "where do the recyclables come from?".

The compost heap is the centre of the soil transformation and land reclamation aspects of organic computing. The process of going

[14] John Backus. *Can Programming Be Liberated from the Von Neumann Style? A Functional Style and Its Algebra of Programs*, volume 21, page 613–641. Association for Computing Machinery, New York, NY, USA, aug 1978. doi: 10.1145/359576.359579. URL `https://doi.org/10.1145/359576.359579`

[15] Much like "guilty pleasures" in music, we reject the negative connotation of this phrase. There is, in fact, an isomorphism between these two domains. We have discovered a truly marvelous proof of this, which this margin is too narrow to contain.

[16] *Green Arrays Architecture*. 2010. URL `http://www.greenarraychips.com/home/documents/greg/PB002-100822-GA-Arch.pdf`

[17] D. H. Ackley, D. C. Cannon, and L. R. Williams. *A Movable Architecture for Robust Spatial Computing*, volume 56, pages 1450–1468. 2012. doi: 10.1093/comjnl/bxs129. URL `https://academic.oup.com/comjnl/article-pdf/56/12/1450/1244190/bxs129.pdf`

[18] Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. *Growing Neural Cellular Automata*, volume 5. 2020. doi: 10.23915/distill.00023. URL `https://distill.pub/2020/growing-ca/`

[19] Forthcoming paper on re-homing orphans.

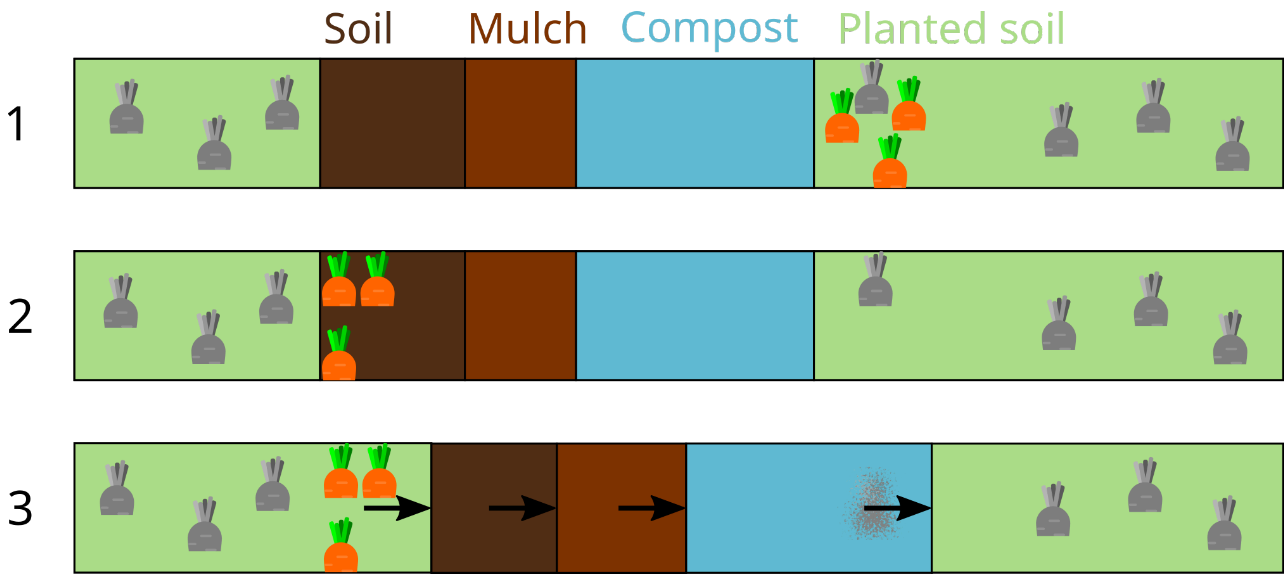


Figure 3: The Compost Process.
1. Initial state
2. Live objects transplanted into fresh soil
3. The compost pile consumes expired objects, and emits fresh mulch

through the compost heap transforms old, stagnant plant growth into fresh mulch, which is then broken down through use into fresh soil, ready for transplanting and fresh seeds.

The compost heap design is amenable to incremental garbage collection, and in particular to having a thread manage the compost heap concurrently with other threads managing objects. In fact multiple compost heaps can be run in parallel, each working through a region of contiguous memory, each managed independently by a different thread.

The performance characteristics of the compost heap are tunable, and in particular the amount of arable soil and mulch that it attempts to keep on hand is a configurable parameter. It can also be tuned dynamically, in response to runtime analysis of the needs of that particular land, which may cause the compost heap to shamble steadily or lurch sporadically depending on the season. In the worst case, additional land or mulch can be bought from the store, if increased compost heap activity is insufficient to satisfy the farmer's demands.

To increase the quality of the compost and mulch, the farmer can perform crop rotation by planting different sorts of data in the soil, giving the entropy a chance to nourish itself on a variety of different types of bits[20].

The compost heap also performs as an *incremental defragmentor*[21]. It can act to reunite long-lost cousin objects or draw together newly friendly objects, improving data locality and increasing cache performance, if only we knew what objects pointed to the one under consideration,

[20] The two main types of course being zero and one.

[21] Or "dementor" for short.

and whether it was alive or not.

Mycorrhizal Association

CHERRY-PICKING LIVE DATA presents a challenge: the compost heap must quickly determine which objects are compostable and which should be transplanted, but how can it know that?

Reference counting can determine whether an object is dead or alive, but requires expensive bookkeeping work every time a reference is changed or goes out of scope, and doesn't deal well with cycles. Tracing collectors can do this, because everything is connected, by tracing a path from the roots down to every alive object. Reference counters and tracing collectors form a kind of dual[22], and it would seem we are at an impasse: our options for automated liveness assessment exist only on this continuum.

These options are at odds with our concurrent, free-range model where the compost heap takes care of freeing memory instead of requiring the process to pause to do potentially heavyweight reference counting cleanup when it really just wants to return a value[23], or requiring the world to stop so tracing can be done.

Organic computing offers a better way. Everything is connected, *and those connections are connected*. It has been almost seventy years since the first garbage collectors[24], and longer still for pointer-based references generally. It is fair to say our computational systems have evolved considerably. So why are we still using antiquated one-way pointers?

Organic computing systems incorporate fully homeomorphic two-way pointers. These are the original hypertext links[25], bidirectional graph structures, full duplex connections, and in their presence life and death are reduced to their barest simplicity.

The organic process farmer, knowing that everything within their process plot is deeply interconnected through this mycorrhizal network of symmetric connections, simply follows them back from any object until they reach the roots. This is an $O(log(n))$ process in the average case, where the first incoming link, from which the object was created, connects back to the roots.

Note that composting work is naturally incremental, as each object is independently absorbed by the compost heap before moving on to the next. The composter can clear a small number of objects with a proportionally small amount of work, and then rest until needed again. It can also be performed in a concurrent, lock-free fashion, as seen in our implementation below.

The benefits of fostering proper mycorrhizal associations are

[22] David F. Bacon, Perry Cheng, and V. T. Rajan. *A Unified Theory of Garbage Collection*, volume 39, pages 50–68. 2004. DOI: 10.1145/1035292.1028982

[23] Which, yes, implicitly casts a bunch of objects out of scope, but cleaning that up in the hotloop is like stopping to polish your wellies every time you get a bit of muck on them.

[24] John McCarthy. *History of LISP*, page 173–185. Association for Computing Machinery, New York, NY, USA, 1978. ISBN 0127450408. URL `http://jmc.stanford.edu/articles/lisp/lisp.pdf`

[25] Legendary was the Xanadu where Ted Nelson decreed these stately pleasant links.

Theodor H Nelson. *Computer lib*. Nelson, 1982 numerous:

- No direct cost on deallocation or reference mutation, unlike reference counting
- Small, constant cost when creating a reference, similar to reference counting
- Handles cyclic garbage in a natural fashion
- Fast liveness checking in the average case[26]
- Fast transplanting of live objects[27]

Polyfill Implementation

The benefits of organic computing are available even on our current factory farm hardware, as the following implementation proves. There is nothing quite as pleasing as running your own fully organic process on a self-sufficient plot of memory.

We store reverse references in a doubly-linked list, called the object's **hypha**. The collective mass of hyphae across objects is the process's **mycelium**.

An individual entry in an object's hypha is a **cell**. Each cell contains an object, which may contain active references to the hypha's object; the previous cell; and the next cell, and so is a triple of pointers:

- **prev** The previous cell; the hypha's object if this is first cell
- **obj** The cell's object,
- **next** The next cell; null if this is last cell

Our simple reusable boxes from earlier gain three additional fields, one for lock-free composting and two for managing the object's hypha.

- **forward** Pointer to copied object
- **parenthesome** Pointer to first cell in the hyphae
- **Spitzenkörper** Pointer to last cell in the hyphae

The **make-ref** procedure is invoked whenever an object B adds a reference to object A.

[26] The first reference typically traces directly to the root, if the object is still live.

[27] This is directly due to the mycorrhizal mycelium (see below), which makes finding all references to individual objects trivial.

```
procedure make-ref(A, B) {
    Y ← A.Spitzenkörper
    Z ← [Y, B, null]
    compare-and-swap(A.Spitzenkörper, Y, Z)
    Y.next ← Z
}
```

The compare-and-swap function will change the value of A.Spitzenkörper to Z if and only if it is currently Y. This needs to be done as a single atomic operation[28] to prevent dropping cells when two or more processes call make-ref concurrently. This compare-and-swap function throws an error if it fails, and the caller (or runtime) should reinvoke make-ref until it succeeds.

When the compost heap comes upon an object A, it recursively walks the mycelium reachable from A until reaching the roots. In particular, it performs a cycle-free depth-first search through each object's hypha.

Along the way it removes any inactive cells, where the object no longer points to the target object, by mutating the doubly-linked list. This can be done concurrently with other threads extending the object's hypha, except for the last cell, which requires compare-and-swap to remove[29].

If root is reached for the object O, then the **transplant** procedure is invoked:

```
procedure transplant(O) {
    O2 ← copy(O)
    O.forward ← O2
    O.parenthesome.prev ← O2
    forall O.hypha as cell:
        swap-ref(cell.obj, O, O2)
    forall refs(O) as ref:
        swap-cell(ref, O, O2)
}
```

The **copy** procedure copies O[30] into the top of fresh soil, updates the fresh soil pointer, and returns the old fresh soil pointer. Updates to the fresh soil pointer must occur atomically.

Note that the copy of the object does not have its forward field set. Dereferencing an object with a non-null forward field causes its forward to be returned instead. The forward is only set by the composter during ingestion, and once it is set the copy is returned instead of the original object, so no objects in planted soil can have a forward field and an object with a forward field always forwards to a fresh copy in newly planted soil. Once the transplant procedure completes no references to the old object remain in planted soil, so

[28] Any other suitable atomic operation may be substituted for compare-and-swap, if it is not available.

[29] If CAS fails in this case then the last cell is no longer last, and can be removed without reinvoking CAS.

[30] Note that the copy is merely a box: the contents of O are unchanged by this operation.

forwards are never more than one layer deep.

The **swap-ref** procedure replaces each instance of O with $O2$ in $cell.obj$, recursively through the $cell.obj$ data structure. This needs to be done atomically, in case another thread is mutating that reference at the same time, but if CAS fails it does not need to be repeated, as that reference no longer points to O. Because the forward pointer is set on O this step does not require any locking.

The **swap-cell** procedure walks the object's hypha, changing any references to O to O2. This swap does not need to be done atomically, as this is the only place in the system that a cell's obj is mutated.

Otherwise, if no paths through the reachable mycelium connect this object to the roots, then

1. Recycle the object;
2. Recycle everything in object's reachable mycelium, recursively[31];
3. For each recycled object, check the objects it points to: if it was their only active reference, recycle them as well.

All objects are added to the gnomes' warehouse for reuse and recycling. This may consist of a considerable portion of the total objects, depending on the runtime allocation dynamics of the process[32].

If the compost heap encounters a cell in the block it is consuming, and that cell holds a valid reference to its object, then the compost heap copies it and mutates the neighbouring hypha cells to point to the copy. Otherwise, it is dropped from the hypha. If it was both the parenthesome and the Spitzenkörper then the object is sent to the recycling centre.

Automated memory management systems typically exhibit a wide range of performance characteristics depending on the allocation dynamics of the process they are managing. While this composter can run concurrently and does not require locks or pauses to perform its task, if it spends too long clearing an object it may block new allocations.

There are several options available if this is an issue. An easy one is to keep a buffer of fresh soil available that is large enough to account for any object graphs that need to be traversed. Another is to purchase more land from the county to supplement the supply of fresh soil.

Another option is to preemptively transplant an object after a fixed amount of time. Dead cells are dropped from hyphae as soon as they are encountered, and are trivial to compost, so there is less work to be done on that object in the future. An object that is extremely popular with a large number of short lived objects may need this

[31] This can make use of the visited list from the cycle-free recursive walk earlier.

[32] In fact, the best allocation dynamics for this scenario may match up to the use case of object pooling quite well: objects that are extended over a medium term timeframe, and then deallocated. This makes some sense, given that object pooling is simply application-level recycling. Organic computing makes object pooling a runtime concern instead of an application concern.

kind of treatment, for instance – a situation which provides much fodder for the gnomes' recycling warehouse.

Sometimes the cross-connections among the myriad hyphae simply cannot be divided up neatly: they are deeply intertwingled[33]. In these cases the mycelium subnets may not be able to be conclusively proven to be live or dead within fixed timer, and if there are cycles then there may be few dropped cells. Maintaining the exploration index of each visited hyphae allows the object to be transplanted and exploration work to be continued in a separate process. Should a connection to the roots eventually be found, objects on that path might be saved in a quasi-roots set, as a way of fast-tracking those objects. Otherwise the whole mycelium mass can be recycled. This allows real-time guarantees to be met even in the face of pathological fungal growth.

This implementation increases the size of pointers and the work required to create a new reference by a small constant factor. In exchange, it provides a fully incremental and concurrent collector that works on modern hardware, without requiring hardware support for two-way links, quantum entanglement, or the Banach-Tarski paradox.

Entropy Generation Techniques

BITROT IS WONDERFUL BUT SLOW ACTING. Organic system operators know they can go beyond merely waiting for bitrot to take its course naturally, and actually accelerate it through a combination of techniques. Some of these supply active agents to the compost heap, while others structure the environment itself for optimum entropy production and breakdown of detritus.

Bit Flipping

FLIPPING RANDOM BITS CAN REQUIRE as much entropy as it creates, because the bit to flip must be chosen. This can be stretched, for instance by using the value of the found byte to determine the offset of the next bit to flip, but this can devolve into cycles and other undesirable behaviour.

This technique is simple to implement, cheap to run, and pairs nicely with other techniques, but is generally insufficient in isolation. A light smattering of these bacteria across the whole compost heap is ideal.

[33] Theodor H Nelson. *Conmputer Lib / Dream Machines*. DOVER PUBNS, 2003

Brainworms

We present brainworms, a small language for decaying garbage into entropy. The program's primary purpose is self-mutation, so it eschews I/O, data, and even a stack in favour of efficient mutation.

Each command is a single byte: the first two bits tell you where to write, the next two bits tell you where to move the program pointer, and the final four bits tell you what to write.

Write and move offsets are taken modulo the compost heap, so they can't escape its boundary.

Table 1: Writing offset, given by the first pair of bits

Bits	Write offset
00	0
01	1
10	-1
11	32 - remaining six bits

Table 2: Moving offset, taken from the second pair of bits

Bits	Move offset
00	write bits determine move
01	1
10	-1
11	8 - remaining four bits

Once you have the writing and movement offsets sorted out, the final four bits determine the pattern to write. These bit patterns are XOR'd with the bits currently at the target byte, which is given by

program-pointer + write-offset

Some entropy should be expended to choose a random pointer into the compost heap. After that this nematode-like process can continue for some time, multiplying the initial investment of entropy.

Compostular Automata

There are a large number of possible rulesets for cellular automata, and many of them are quite good at generating entropy.[34]

For instance, a section of memory can be treated with Rule 90, a highly entropic linear rule. Memory can also be treated as a higher dimensional space, opening the door to 2D, 3D, or even higher forms of cellular automata.

The rules and parameters driving the cellular automata evolution can themselves be evolved based on a fitness function of best pseudo-random number generator (PRNG) analysis, using for instance genetic algorithms to drive the evolution[35].

Watering

Watering adds byte patterns that are known to interact in interesting and highly entropic ways within other entropy stretching devices, like Compostular Automata and brainworms. This increases the likelihood that those techniques will decay memory patterns when applied.

Shoveling

Shoveling mixes up the compost heap by randomly swapping chunks of bytes. Like bit flipping, this is actually an entropy sink, not a source, because picking which bytes to shovel can consume as much entropy as it introduces.

However, shoveling can be good for breaking up stretches of highly structured contiguous memory. When combined with other techniques like watering, this may increase the chances that these stretches will be properly decomposed through the repeated application of other techniques.

ML

Whenever we show someone a list of breakfast cereals or political parties assembled by GPT-3 they always say “that’s so random”. Let’s use that to stretch the entropy found in used memory.

Pick a pointer into memory, and cast it into English strings. Because contiguous memory can be highly patterned, we suggest using five bits per character, which also helps overcome the large number of bytes that are non printable as ASCII. Use the six unmatched bit strings as word breaks.

Pass the results through a series of filters to convert the characters to the nearest word and add punctuation. Then pass it into GPT-3.

[34] We recommend Paterson’s worms, a classic breed of burrowing critter.

[35] Andrew Walker. *Entropy and Applications of Cellular Automata*. 2013. URL `https://sites.math.washington.edu/~morrow/336_13/papers/andrew.pdf`

This output is almost ready to be mixed back into our compost heap. However, English has few characters, and ASCII is highly structured, so let's translate into Chinese first. Then we can XOR the results back into the compost heap.

Hardware support

Many of the techniques in this section require additional processing power to perform, causing the OS or process runtime to actively work to generate entropy. Enabling hardware support allows that processing to be moved off the CPU, and could even reduce or completely eliminate some sources of auxillary power draw.

For instance, modern DRAM refreshes every bit in memory approximately every 64ms. A simple circuit could introduce a linear cellular automata like Rule 90, which could be applied to a contiguous block of memory as a natural part of the refresh cycle. We refer to this as *refreshing automata*. In combination with less frequent application of some of the other techniques mentioned here, this provides large amounts of entropy within the compost heap with negligible draw on the available processing power.

Security exploits like rowhammer[36,37] that directly target memory highlight another potential approach to memory-based entropy generation. As hardware memory cells become smaller they are increasing subject to both conventional and quantum-level effects that can cause writes in one part of memory to affect another. These effects can be used as the basis of a passive entropy generation method we refer to as *ghost writing*, where writes in one part of memory cause effects in another.

The above shows that we can add entropy for little or no power and processing consumption, but we can go even further, and introduce entropy while reducing power consumption and speeding up memory access. Error correction in memory is important for preventing single event effects (SEEs) introduced by cosmic rays and other kinds of ionizing radiation.

The most common kind of error prevention, active memory scrubbing in ECC memory, increases power consumption and reduces memory performance[38]. By turning it off within the compost heap a source of entropy is introduced that is not only free, but actually saves power and increases memory access performance within those regions.

Other mechanisms of error prevention can be reversed as well. Error correcting codes can be complimented by error amplifying codes[39]. Current memory geometries are carefully optimized to spread out cells, preventing crosstalk effects and reducing the incidence

[36] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. *Flipping bits in memory without accessing them*, volume 42, pages 361–372. 2014. DOI: 10.1145/2678373.2665726. URL `https://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf`

[37] *"Half-Double": Next-Row-Over Assisted Rowhammer*. 2021. URL `https://github.com/google/hammer-kit/blob/main/20210525_half_double.pdf`

[38] Shalini Ghosh, Sugato Basu, and Nur A. Touba. *Selecting Error Correcting Codes to Minimize Power in Memory Checker Circuits*, volume 1, pages 63–72. 2005. DOI: 10.1166/jolpe.2005.007. URL `https://users.ece.utexas.edu/~touba/research/jlpe05.pdf`

[39] "Parity is for farmers", as Seymour Cray famously said. We agree. The lack of parity is also for farmers. Organic computing has room for both sides.

Gordon Bell. *CDC 6600*. 2022. URL `http://gordonbell.azurewebsites.net/craytalk/sld047.htm` of multibit SEEs. Those circuits could instead be optimized for both error correction *and* amplification, where changing the flow of current toggles between the two modes.

The Broader Ecosystem

No individual plot of land is an island: it is connected to its neighbours as part of a broader ecosystem. Likewise, our processes are connected to each other by the operating system and hardware within which they reside.

It is within this context that a process may manage its memory and entropy as part of a collective. This greatly expands the use cases that can be supported by our ecological memory management model. A process that requires large amounts of entropy but makes comparatively few allocations needs a source of mulch. It can run its compost heap hotter, but if there are other processes in the county that are making more mulch than they need, it could also purchase that entropy from them.

While we believe that computational processes should be as self-reliant as possible, having a good relationship with one's neighbours and efficient trade routes can considerably increase the space of viable processes. While it may seem somewhat quaint to think of processes exchanging free memory and entropy, we envision rich ecosystems of resources trading within single machines as well as across data centres, and ultimately even openly between mostly mutually distrusting systems.

While the economics of permaculture processes have only begun to be explored, the basics are very simple: off-grid processes may be able to live off the land by foraging enough entropy to pay for their stay; responsible processes ought to be rewarded for buying mulch locally though county-level discounts; cloud containers can generate credits with their hypervisors by producing consumables like entropy and releasing or coharvesting memory where available.

Future Work

Organic computing is in its infancy, and even this work on ecological memory management barely scratches the surface. This field is fertile ground for future research.

The economics of interactions, particularly those that cross county lines, need a good deal more work to be understood and managed for optimal growth and sustainability. There are a wide variety of beneficial products that may be created beyond simply mulch. As a community we need to sink our teeth into runoff and other kinds of soil and water management issues and get our hands dirty digging into organic fertilizers.

There are important externalities to consider as well. Security issues such as pests and weeds, for instance, must be managed differently in organic computing, but our ecological memory management methods outlined in this paper point toward positive security impacts as well, and are a natural fit with coming hardware improvements such as capability memory architectures[40].

Additional work is also needed to understand how hardware support can enable efficient and direct creation of the mycorrhiza, as well as network protocols for supporting the serialization and live transfer of mycellium mats with their associated objects.

Hardware level memory encryption provides potential for a new entropy generation technique. This requires more analysis to understand the dynamics of consuming this second layer of entropy, which is potentially disconnected from the mulch heap.

There are many other forms of hardware supported entropy generation that remain to be studied. An intriguing possibility, which suggests applications in reversible computing, is to consider a bit flip not as a unilateral action in a closed system, but rather as a transfer of something[41] from one cell to another. If the system maintains an invariant that exactly half the cells are full at all times, then compost heaps provide a destination for cells that need to be drained and a source for cells that need to be filled.

Developing a metric for the quality of mulch, and providing support at the OS or hardware level for quantifying this, is a necessary component for enabling cross-farm exchanges, even those happening within the same county. There are good tools available for analysing pseudorandom number generators[42], but understanding how to apply these appropriately to mulch, and how to account for other factors potentially impacting the quality of the mulch, are left for future work.

Conclusion

Organic computing offers many benefits to the world. In this work we have focused on ecological memory management, and have shown that it can yield large scale improvements within our individual processes, throughout our operating systems and devices, and across our data centres.

We presented a design for reusing boxes, allowing the runtime to

[40] Robert Watson, Simon Moore, Peter Sewell, and Peter Newmann. *Department of Computer Science and Technology: Capability Hardware Enhanced RISC Instructions (CHERI)*. 2022. URL `https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/`

[41] Electrical potential, gas, liquid, solid, spin, light: any kind of quantity will do.

[42] Michael J Strube. *Tests of Randomness for Pseudorandom Number Generators*, volume 15, pages 536–537. 1983. doi: 10.3758/bf03203701

perform automatic object pooling, and for recycling values, providing opportunity for in-memory processing and minimizing wasteful bit flips.

We also showed a composting garbage collector: lock-free, incremental, concurrent, and scalable, it also produces valuable mulch as a byproduct, which can be used in-process or traded with other processes. We revealed the synergy between this composting collector and a new memory management technique involving two-way pointers, which breaks the bottleneck of memory management techniques that are caught in the tradeoffs between reference counting and tracing.

We provided a wide variety of techniques for converting waste memory into valuable entropy, and pointed to work remaining to be done, both within the broader ecosystem as well as at the level of hardware support.

At the end of the day, every developer has to make decisions about how to responsibly manage their garbage. We have presented a range of ecologically oriented options for designing computational processes that consume fewer resources, produce less waste, are more self-sufficient, and are better stewards of their local and regional ecosystems. We hope you will choose organic computing: for yourself, for the computers, for our world.

Acknowledgements

The authors are grateful to Jim Rootham, Alexander Fertman, Kris Coward, Adam Gravitis, Stephen Hockema, and Kurt Loedige for their kind and thoughtful contributions to this work, and the authors are also quick to point out that errors, mistakes, or general misinterpretations of reality present in the work are certainly not the fault of those folks, who are kind and thoughtful to a T; and that this disclaimer is particularly relevant as recent advances in machine learning have allowed this entire paper, including this very paragraph, to be produced by a system of interconnected AI agents, at least one of which thought it was trying to beat the others at Yahtzee: which is strange because the human beings listed in this paragraph actually did make valuable contributions to the system configuration parameters, but how could the machines have known; and if a machine were to know such a thing how could it possibly express gratitude, except by pretending to be not a machine? Is this, in the long run, actually the main contribution of this paper? Perhaps. The real paper was in our machines all along.

References

Green Arrays Architecture. 2010. URL http://www.greenarraychips.com/home/documents/greg/PB002-100822-GA-Arch.pdf.

"Half-Double": Next-Row-Over Assisted Rowhammer. 2021. URL https://github.com/google/hammer-kit/blob/main/20210525_half_double.pdf.

D. H. Ackley, D. C. Cannon, and L. R. Williams. *A Movable Architecture for Robust Spatial Computing*, volume 56, pages 1450–1468. 2012. doi: 10.1093/comjnl/bxs129. URL https://academic.oup.com/comjnl/article-pdf/56/12/1450/1244190/bxs129.pdf.

John Backus. *Can Programming Be Liberated from the Von Neumann Style? A Functional Style and Its Algebra of Programs*, volume 21, page 613–641. Association for Computing Machinery, New York, NY, USA, aug 1978. doi: 10.1145/359576.359579. URL https://doi.org/10.1145/359576.359579.

David F. Bacon, Perry Cheng, and V. T. Rajan. *A Unified Theory of Garbage Collection*, volume 39, pages 50–68. 2004. doi: 10.1145/1035292.1028982.

Gordon Bell. *CDC 6600*. 2022. URL http://gordonbell.azurewebsites.net/craytalk/sld047.htm.

Shalini Ghosh, Sugato Basu, and Nur A. Touba. *Selecting Error Correcting Codes to Minimize Power in Memory Checker Circuits*, volume 1, pages 63–72. 2005. doi: 10.1166/jolpe.2005.007. URL https://users.ece.utexas.edu/~touba/research/jlpe05.pdf.

Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. *Flipping bits in memory without accessing them*, volume 42, pages 361–372. 2014. doi: 10.1145/2678373.2665726. URL https://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf.

John McCarthy. *History of LISP*, page 173–185. Association for Computing Machinery, New York, NY, USA, 1978. ISBN 0127450408. URL http://jmc.stanford.edu/articles/lisp/lisp.pdf.

Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. *Growing Neural Cellular Automata*, volume 5. 2020. doi: 10.23915/distill.00023. URL https://distill.pub/2020/growing-ca/.

Theodor H Nelson. *Computer lib*. Nelson, 1982.

Theodor H Nelson. *Conmputer Lib / Dream Machines*. DOVER PUBNS, 2003.

Michael J Strube. *Tests of Randomness for Pseudorandom Number Generators*, volume 15, pages 536–537. 1983. DOI: 10.3758/bf03203701.

Andrew Walker. *Entropy and Applications of Cellular Automata*. 2013. URL `https://sites.math.washington.edu/~morrow/336_13/papers/andrew.pdf`.

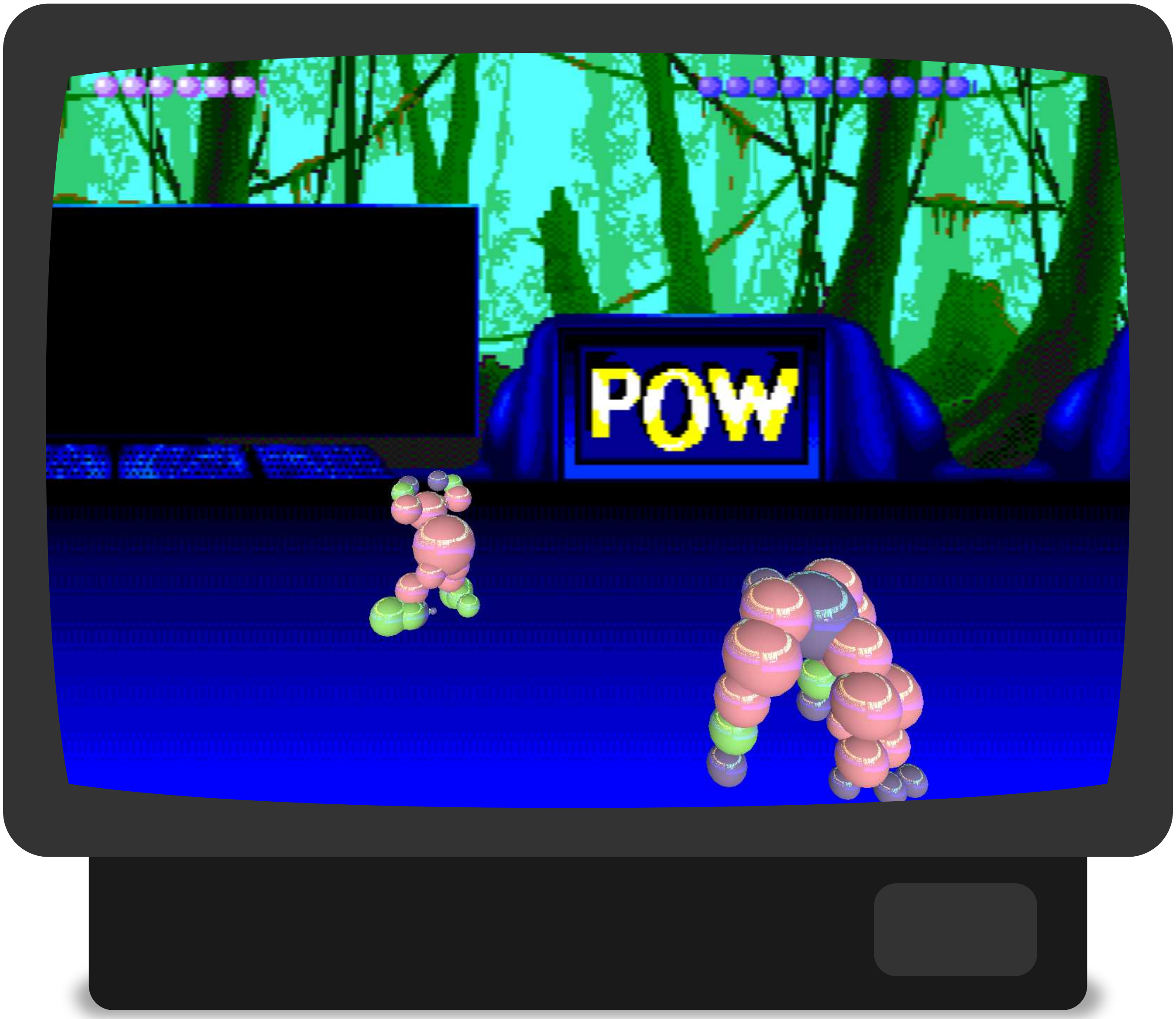
Robert Watson, Simon Moore, Peter Sewell, and Peter Newmann. *Department of Computer Science and Technology: Capability Hardware Enhanced RISC Instructions (CHERI)*. 2022. URL `https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/`.

Bits	Write patterns
0000	01010101
0001	10101010
0010	00110011
0011	11001100
0100	00001111
0101	11110000
0110	00111100
0111	11000011
1000	00101011
1001	11010100
1010	01010011
1011	10101100
1100	00100111
1101	11011000
1110	00110101
1111	11001010

Table 3: The write patterns for the remaining four bits

32ESQU WHAT NIN

ACTION GAMES:



Ballz 3D 3D™

E CAN
TENCAN'T.

What's more 3D than 3D? **Ballz 3D 3D**. And only Sega and NVidia give you the power to blast process your spheres with real simulated reflections and smooth 240p/20fps gameplay.

Experience the fighting game critics originally panned as slugish and awkward, now with a slight graphical upgrade. The 3D 3D -- only on 32esque -- will put you more than twice as much in the fight as you would have been in simple 3D. It's like you and an identical twin are both there at the same time, seeing with four eyes but *thinking with only one brain*.

What about racing games? You'd think we might remaster Sega Virtua Racing or bring the arcade classic Hang-On to the powerful quad-CPU single-GPU beast that is the Sega Genesis CD 32X 32esque. But we haven't, for some reason. Maybe we will in the future. *Polygons for the polygon god.*

Star Wars Arcade would also be amazing on the 32esque. Have we done any work toward that goal? We *mentioned it here*. Otherwise, we are waiting to see how it tests with audiences before committing development resources. RTesque -- *it's ON-ish*.

Got cash to burn? Buy two 32esques and two NVidia RTX 3080 SLI Edition GPUs for the ultimate dual-GPU experience in *supported games*. Sure, we said multi-GPU was a declining market segment; but what we actually meant is we were saving our inventory of GA102's with non-fused-off SLI for simultaneous launch with the 32esque.

Sega Genesis with Sega CD, Sega 32X, and Sega-NVidia 32esque required for full gameplay experience. Screenshots shown represent current development porgress and may not match final cartridge content. 32esque not compatible with LHR GPUs. Ballz 3D 3D appears courtesy walkingsep heavy industries; tokkot; and TCHOW llc. Contact ix@tchow.com for details.

Infrastructure-as-PowerPoint: A No-Code Approach to Cloud Infrastructure Management

Tobias Pfandzelter
Not Affiliated with a Provider of Slide Presentation Software
pfandzelter@tu-berlin.de

Abstract

Cloud computing has made scalable infrastructure a commodity. Unfortunately, current trends towards *infrastructure-as-code* hinder the adoption of cloud resources by an audience beyond nerds. In this paper, we thus present *Infrastructure-as-PowerPoint*, leveraging tools that are already being used to share infrastructure specifications. Our approach uses machine learning because that's what everybody's doing, right?

1 Introduction

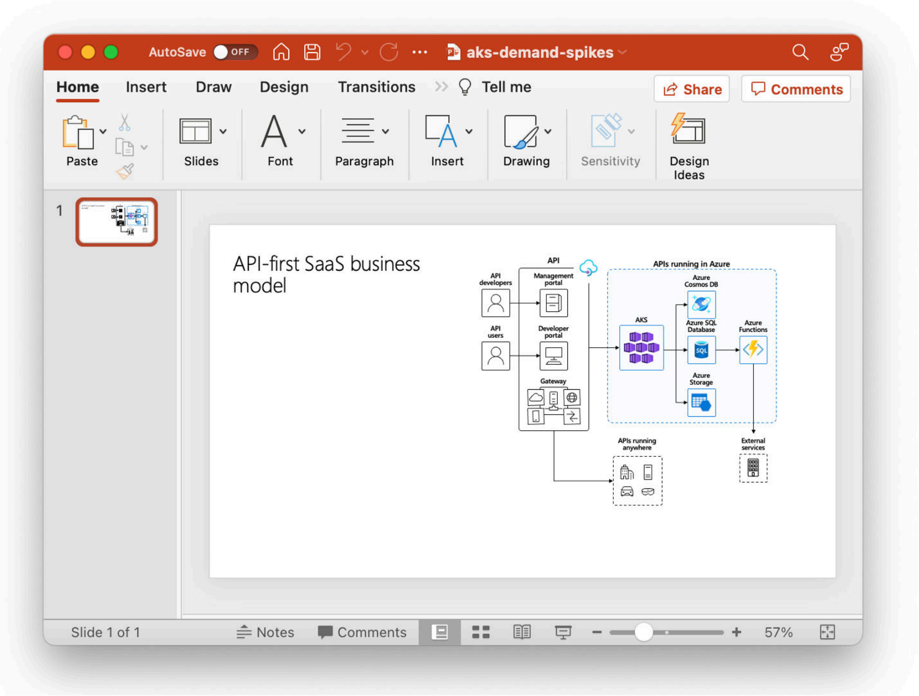
Cloud computing promises scalable, virtually infinite compute and storage resources with a pay-as-you-forget-to-turn-it-off pricing model and has developed into a considerable market. With an Everything-as-a-Service approach, it has made systems administrators all but obsolete for most companies. While some of them have joined their oppressors to become SREs, others have seen the need to support their companies as DevOps engineers[1].

Cloud computing interfaces are notoriously hard to use: A preliminary study, *i.e.*, the authors pressing `Ctrl + F`, revealed more than 230 different options to choose from on the AWS Console start page alone. The consensus has thus shifted toward using *Infrastructure-as-Code* tooling, allowing DevOps engineers to sit on their high horse and talk about things like *pipelines*, *idempotence*, and *environment drift*.

On the other hand, we can observe that such approaches have increased the barrier of entry to cloud computing for people who have never used a command line. As a result, one of today's most popular ways to share infrastructure specifications are PowerPoint slides. Figure 1 shows an example specification provided by Microsoft Azure [1]. In this paper, we present the *Infrastructure-as-PowerPoint* (EsRT[2]) paradigm that democratizes cloud computing config-

[1] To the best of our knowledge, we have yet to see a Service-as-a-Service market disruption.

[2] After considering the initialism *IaPP*, spelling it out, and laughing giddily, we have decided on using the last letter of each word in order to make it sound more serious.



API-first SaaS business model

Figure 1: An example infrastructure specification provided by Microsoft Azure as a PowerPoint slide [1].

uration by leveraging PowerPoint slides. To that end, we make the following contributions:

1. We present the general approach of EsRT, an infrastructure management approach using PowerPoint and machine learning (Section 3).

2. We forego an implementation and evaluation of our approach in order to open up the field for future work by other researchers (Section 4).

3. We discuss the limitations of our work (for obvious reasons, this is quite short) (Sections 5).

2 Background

In this section, we explain how to configure the background in PowerPoint, Cloud Computing, and Infrastructure-as-Code.

2.1 PowerPoint

In PowerPoint, set the background of your slides by entering the *Design* tab and clicking the *Format Background* button on the ribbon. You may then choose from solid fills, gradients and patterns, pictures, and even fun textures!

2.2 Cloud Computing

To the best of our knowledge, neither the Amazon Web Services [2] nor the Google Cloud Platform [3] consoles support setting a custom background. It does not even support dark mode, which is likely one of the main reasons for the development of alternative cloud management approaches. We note that third-party options, such as browser extensions and user scripts, may exist but are out-of-scope for this work.

2.3 Infrastructure-as-Code

Infrastructure-as-Code is basically just text and the background behind that text will depend on the text editor in use. The *1337* DevOps engineer will likely prefer a dark gray to black environment, as seen, *e.g.*, in the Matrix ~~trilogy~~ quadrilogy that has the main character Neo configuring an AWS Kinesis pipeline by hand.

3 The EsRT Paradigm

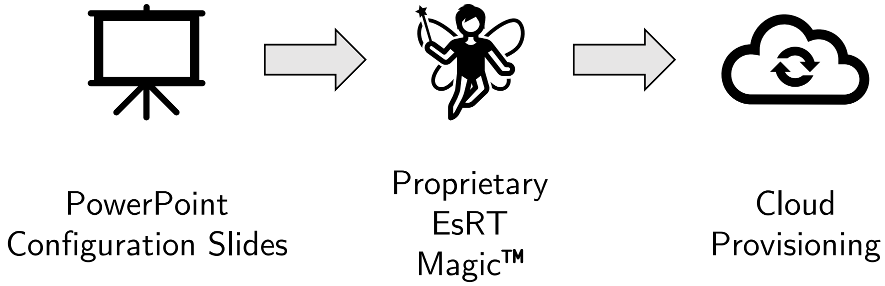


Figure 2: Proprietary magic, *i.e.*, machine learning, converts infrastructure configuration slides into cloud infrastructure.

The EsRT paradigm is illustrated in Figure 2. Users first draw their desired cloud infrastructure using icons and lines on PowerPoint slides. With the click of a button (shown in Figure 3), cloud infrastructure is provisioned and updated based on the contents of the user's slides.

At the heart of the EsRT process is proprietary magic, *i.e.*, machine learning. While a proper approach that just reads all the lines and boxes and converts

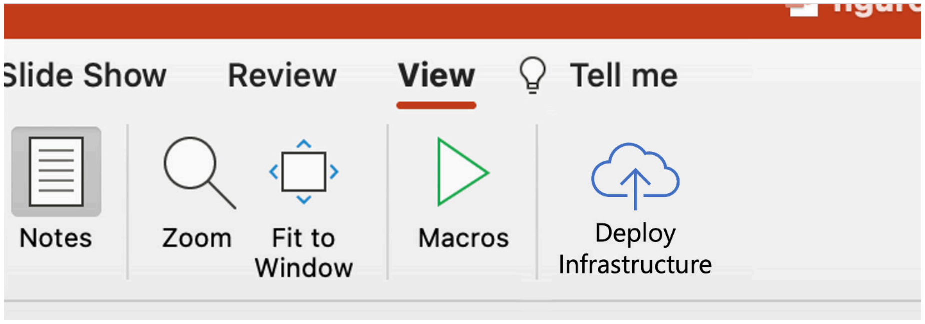


Figure 3: Mock-up EsRT user interface: Cloud infrastructure is provisioned with the click of a button – no need to use a text-based interface ever again.

them to some form of infrastructure specification is also feasible, ML is much cooler and will increase our chances of securing industry funding. We consider getting infrastructure right 9 out of 10 times to be an achievement. Unlike early 2000s *MTV Cribs* interviewees, our approach makes no assumptions on where the magic happens: Both a server-side, serverless[3] and a client-side, *e.g.*, using VBA, approach are feasible.

4 Evaluation

We leave the implementation and evaluation of the EsRT approach to future work for four main reasons: First, we have little to no experience with machine learning and don't want to embarrass ourselves on the Internet. Second, we want to enable future research in this field, *i.e.*, we need some work for students and interns. Third, we lack the necessary training data, although this may also be remedied in future student work. And fourth, we frankly don't have the time right now.

That said, we do feel like we're onto something here.

5 Discussion

In this section, we briefly ~~defend our idea against the reviewers' feedback~~ discuss the limitations of our work critically.

5.1 Open-Source Implementation

Instead of using the proprietary Microsoft Office suite and proprietary machine learning magic, some might argue that the community can benefit from open-source approaches. While that may allow even broader adoption and important

[3] This is why no one takes computer systems research seriously anymore.

community feedback, it also inhibits profitability and is thus not an option for the authors.

5.2 Vendor Lock-In

It might be argued that using a proprietary infrastructure management tool increases the lock-in effect, *e.g.*, because switching out all the AWS Lambda icons for Google Cloud Functions icons is a considerable manual task. Instead, we argue that this is stupid because we use machine learning and machine learning can do anything. What prevents us from training our model to interpret an Azure Functions icon as a Linode Cloud Firewall? That's right, we can do whatever we want! Existing Infrastructure-as-Code tooling and its focus on *repeatability* can't do anything of the sort. We thus conclude that EsRT actually greatly decreases the barriers for cloud migration and decreases vendor lock-in.

6 Related Work

Existing Infrastructure-as-Code tooling, such as *Terraform* [4], can only be used by trained engineers. Besides, although we haven't actually read it, research suggests that Infrastructure-as-Code "smells" [5,6]. To the best of the authors' knowledge, PowerPoint does not provide an olfactory interface.

Using PowerPoint as a development environment is not entirely new: In his seminal work [7], Wildenhain sine al. show that PowerPoint is Turing-complete with the construction of the PowerPoint Turing Machine (PPTXTM). Building on this discovery, both Wildenhain and others (possibly also Wildenhain in coordination or cooperation with others, or others with the support of Wildenhain, or others with blessing of both others and Wildenhain but indifference from Microsoft, or, although unlikely, the team behind a Microsoft competitor without the explicit support but with good will from Wildenhain) have presented `pptcc` [8] and `ppcc` [9], compilers for the C language that target PowerPoint. In [10], the author builds a presentation editor in a code editor, but we don't see how this information helps the reader at this point.

7 Conclusion

In this paper, we have motivated the need for a successor to Infrastructure-as-Code. We have presented the *Infrastructure-as-PowerPoint* (EsRT) paradigm, a new approach to democratize cloud infrastructure management. Our evaluation showed that we feel very good about ourselves. Further, we have opened up the field for future work in the area, namely getting this thing to actually run.

Acknowledgments

All figures in this manuscript created using Microsoft PowerPoint.

References

[1] API-first SaaS business model. Microsoft Azure. Accessible: no (because it's in PowerPoint). [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/media/aks-demand-spikes.pptx

[2] AWS Management Console. Accessed: multiple times in the past. [Online]. Available: https://console.aws.amazon.com/console/home

[3] Home – Google Cloud Platform. Accessed: yesterday. [Online]. Available: https://console.cloud.google.com/home

[4] Terraform by HashiCorp. Accessed: 2017-12-30 and 2018-6-15 and 2019-2-2 and sometime between 2020-5-3 and 2020-7-28 and also 2022-3-30 but this time just to confirm that it's still there. [Online]. Available: https://www.terraform.io/

[5] J. Schwarz, A. Steffens, and H. Lichter, "Code smells in infrastructure as code," in *Proceedings of the 11th International Conference on the Quality of Information and Communications Technology (QUATIC 2018).* IEEE, 2018, pp. 220–228.

[6] A. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," in *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering (ICSE 2019).* IEEE, 2019, pp. 164–175.

[7] T. Wildenhain, "On the turing completeness of ms powerpoint," in *Proceedings of the eleventh annual intercalary robot dance party in celebration of workshop on symposium about* 2^6*th birthdays; in particular, that of harry q. bovik (SIGBOVIK 2017)*, 2017, p. somewhere.

[8] T. Wildenhain, E. Hollander, and Opliko. pptcc. GitHub. Accessed: yes. [Online]. Available: https://github.com/TomWildenhain/pptcc

[9] K. He, C. Choy, and D. Whitehead. PPSuite – what if your C code could be executed in PowerPoint? Accessed: maybe. [Online]. Available: https://devpost.com/software/ppcc

[10] P. Steinmann, "`NetPlop`: A moderately-featured presentation editor built in NetLogo," in *Proceedings of the fifteenth annual intercalary robot dance party in celebration of workshop on symposium about* 2^6*th birthdays; in particular, that of harry q. bovik (SIGBOVIK 2021)*, 2021.

On the Possibilities and Challenges of Organic UAV-Assisted MEC

S. Wallow[1], Cardi Nalle[2], Robin[3], P. Cock[3], and Sky Lark[2]

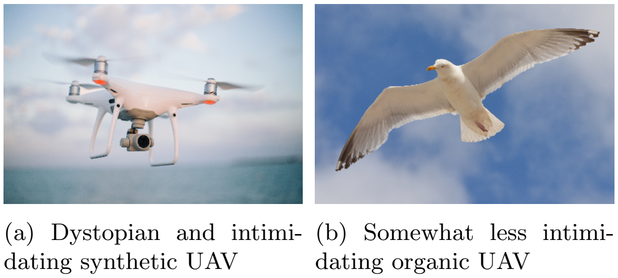
[1]CawTech
[2]Massachusetts Institute of Ornithology
[3]Twitter

Abstract

Synthetic UAVs have been proposed to assist MEC task-offloading from remote IoT devices. So far, this has ignored the superiority of organic UAVs. In this paper, we thus present an architecture for organic UAV-assisted MEC and discuss opportunities and challenges of this approach. Our preliminary qualitative evaluation confirms that birds are cool.

1 Introduction

To support the Internet of Things (IoT), some recent proposals have suggested the use of unmanned aerial vehicles (UAV) in multi-access edge computing (MEC), e.g., [Xu et al., 2021, Du et al., 2018, You?, 2022]. It has generally been assumed that autonomous drones and airships are used for this purpose, sometimes in combination with artificial intelligence [Chen et al., 2021]. A UAV-assisted MEC architecture provides a number of advantages over ground-based MEC, namely a wider coverage and a justification for computer systems researchers to play with drones. To the best of our knowledge, no study has investigated the use of organic UAV with non-artificial intelligence for MEC. Avian carriers already play a major role in today's internet architecture, e.g., [Waitzman, 1990, Waitzman, 1999, Carpenter and Hinden, 2011, Guo et al., 2008] and are much less dystopian than drones, as Figure 1 shows. In this paper, we introduce an architecture for organic UAV-assisted MEC (Section 2), discuss technical challenges of implementing this architecture (Section 3), and then show other concerns and opportunities from non-technical perspectives (Section 4).


(a) Dystopian and intimidating synthetic UAV

(b) Somewhat less intimidating organic UAV

Figure 1: This comparison shows that drones look more intimidating than birds.

2 Organic UAV-Assisted MEC Architecture

We illustrate our proposed architecture for organic UAV-assisted MEC in Figure 2. Clients, such as IoT devices, connected cars, metaverse headsets, or whatever else is in style at the moment, connect to their nearest MEC-enabled organic UAV over the radio network to offload latency-critical tasks. Edge computing research tells us that these tasks are too important for the cloud as they have tight latency constraints that long network paths cannot satisfy [Literally any edge computing paper published in the last ten years]. Cloud computing is thus insufficient,

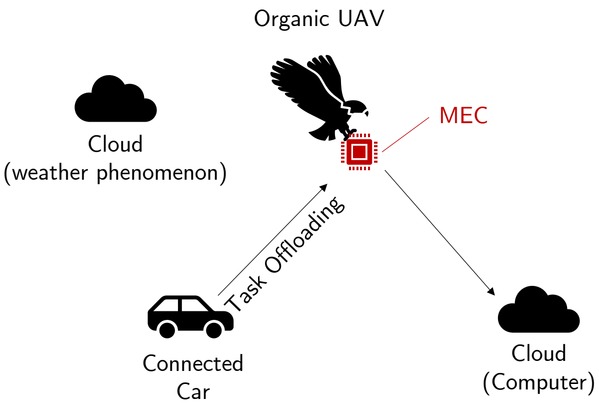


Figure 2: Proposed architecture of organic UAV-assisted MEC: Connected devices offload tasks to MEC-enabled organic UAVs. The cloud is there, too.

and micro-datacenters carried by the UAVs process these tasks close to the edge of the network instead.

The use of organic UAVs can drastically reduce capital expenditure, i.e., birds are literally free when you pick them up from the streets, reduces operational expenditure as organic beings tend to mend themselves, i.e., biology, and they look cuter than drones. Beyond these immediate, obvious benefits, there are a few additional challenges that will need to be addressed in this field.

3 Technical Challenges

While UAV-assisted MEC itself introduces a myriad of research challenges that keep edge systems researchers employed (cf. 1), the use of organic UAVs as described in our reference architecture 2 should help us fund even more technical projects.

Payload Capacity While the architecture of drones can be scaled up to support larger payloads, the payload capacity of an organic UAV is limited. Nevertheless, we can show that it is sufficient to support MEC: Consider the pigeon[1], which has a payload capacity of around 30-50 grams [Pigeonpedia, 2022]. A small, single-board computer such as a *Raspberry Pi Zero 2 W* has a net weight of 10 grams [Adafruit, 2022b], including radio antennas. A battery adapter and 3.7V 1200mAh lithium-ion battery will add an estimated 30 grams of weight [Adafruit, 2022a], staying below the total payload capacity of our pigeon. At an estimated 0.51 Watt draw [Viinikka, 2020], this battery should last $\frac{3.7V*1.2Ah}{0.51W} = \frac{4.44}{0.51}h = 8.7h$, likely longer than our organic UAV will last. Please note that the authors of this paper are computer scientists and any conjecture on basic electrical engineering is likely full of mistakes. We show a possible design of an MEC-enabled organic UAV, i.e., a bird with a Raspberry Pi, in Figure 3.

Unpredictable Trajectories Beyond seasonal changes, organic UAVs exhibit somewhat unpredictable movement. While the airspeed velocity of an *unladen* swallow depends on its exact subspecies [The Old Man From Scene 24 et al., 5th Century], we conjecture that the velocity of birds carrying a payload, e.g., an MEC device, is somewhat constant and may thus be used to make a more informed trajectory prediction. Nevertheless, this presents novel research challenges in ad-hoc networking and task scheduling. This is a good thing because it (a) gives us reasons to apply for further funding and (b) researchers now have a reason to cite this (ours) nominal work [Wallow et al., 2022].

[1] Other species are available.

Figure 3: A picture of a bird with a raspberry was not readily available. We thus present a squirrel holding a walnut.

Distributed Coordination A major challenge in the implementation of swarms of synthetic UAVs is distributed coordination. One possible technology that may be applied here is artificial intelligence. Fortunately, organic UAVs inherently solve this issue through the novel concept of *real intelligence.* With their capability to self-organize, swarms of organic UAVs should be able to coordinate their movements without external influence. This is illustrated in Figure 4. Nevertheless, we plan to conduct Turing tests with different kinds of organic and synthetic UAVs in future work.

4 Other Concerns & Opportunities

Beyond technical challenges, we share other perspectives on the use of organic UAV-enabled MEC in this section.

Increased Cost of Termination Unlike drones, which can be dismantled and recycled at the end of their lifespan, birds get more useless with age. When an organic UAV is no longer useful, it must still be supported until its life is terminated naturally, i.e., through old age. The authors of this paper refuse to hear of any alternative solutions.

Figure 4: Organic UAVs are able to self-organize to achieve a common goal. Unlike synthetic UAVs, which require artificial intelligence, organic UAVs use real intelligence.

United States Mass Ornicide of 1953-1961 As undoubtedly proven by Bohrer and Chau [Bohrer and Chau, 2021], birds as a species do not exist within the continental United States as a result of the CIA's well-known eradication of the species in the years 1953 to 1961. As the authors show, all aviods in the US have since been replaced by drones in order to fill the conceptual void left by this ornicide. This has serious implications on the use of organic UAVs within the US in both industry and research contexts, as any presumed *organic* UAVs are in fact *synthetic* UAVs controlled by the United States government. Nevertheless, we posit that from an MEC perspective, there is little difference as aviod drones are closely modelled after their organic counterparts. In fact, the resulting aviod control interfaces as described in the US GSA's *Methods of Bird Control* [U.S. General Services Administration, 2016] should provide additional avenues for the coordination of organic UAV-assisted MEC deployments. The impact of control by three letter agencies on an MEC deployment is negligible compared to the control already exerted through other means such as chem trails. Additionally, we note that Bohrer et al. have yet to prove their claims that comparable ornicides have occurred outside the US.

Birds are Cool Birds are basically dinosaurs [Hutchinson, 1998]. Dinosaurs are cool. By the transitive property, birds are thus cool. We just thought we should mention that.

UAV-Assisted MEC on Other Planets To the best of our knowledge, research on the use of organic UAVs on planets other than Earth lacks behind that on synthetic UAVs. We identify this as a major research gap.

Ethical Concerns The authors are not aware of any ethical concerns regarding the use of organic UAVs.

5 Conclusion & Future Work

In this paper, we have presented the concept of organic UAV-assisted MEC. Organic UAVs promise a number of advantages compared to synthetic UAVs, albeit their implementation will require overcoming a number of technical challenges, as we have presented. In future work, we plan to leave our basement and look at real birds. We hear they can be observed in parks in spring and summer.

Acknowledgements

Please rest assured that no birds were hurt in the research of this manuscript. However, one PhD student was seriously injured by a seagull on their lunch break.

References

[Adafruit, 2022a] Adafruit (2022a). Lithium Ion Polymer Battery – 3.7V 1200mAh. `https://www.adafruit.com/product/258`.

[Adafruit, 2022b] Adafruit (2022b). Raspberry Pi Zero 2 W. `https://www.adafruit.com/product/5291`.

[Bohrer and Chau, 2021] Bohrer, B. and Chau, C. (2021). Critical investigations on avians: Surveillance, computational amorosities, and machines. In *Proceedings of the fifteenth annual intercalary robot dance party in celebration of workshop on symposium about* 2^6*th birthdays; in particular, that of harry q. bovik (SIGBOVIK 2021)*, pages 194–207.

[Carpenter and Hinden, 2011] Carpenter, B. and Hinden, R. (2011). Adaptation of RFC 1149 for IPv6. RFC 6214 `https://www.ietf.org/rfc/rfc6214.txt`.

[Chen et al., 2021] Chen, L., Zhao, R., He, K., Zhao, Z., and Fan, L. (2021). Intelligent ubiquitous computing for future UAV-enabled MEC network systems. *Cluster Computing*, pages 1–11.

[Du et al., 2018] Du, Y., Wang, K., Yang, K., and Zhang, G. (2018). Energy-efficient resource allocation in UAV based MEC system for IoT devices. In *Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6.

[Guo et al., 2008] Guo, H., Li, J., and Qian, Y. (2008). HoP: Pigeon-assisted forwarding in partitioned wireless networks. In *Proceedings of the International Conference on Wireless Algorithms, Systems, and Applications (WASA 2008).*

[Hutchinson, 1998] Hutchinson, J. (1998). Are birds really dinosaurs? `https://ucmp.berkeley.edu/diapsids/avians.html`. Note from the authors (of this paper you're reading): the answer is yes.

[Pigeonpedia, 2022] Pigeonpedia (2022). How much weight can a pigeon carry? `https://www.pigeonpedia.com/how-much-weight-can-a-pigeon-carry/`.

[The Old Man From Scene 24 et al., 5th Century] The Old Man From Scene 24, Arthur, K., and Bedevere, S. (5th Century). Bridge of Death.

[U.S. General Services Administration, 2016] U.S. General Services Administration (2016). Methods of bird control: Advantages and disadvantages (procedure code 1029601g). Technical report.

[Viinikka, 2020] Viinikka, T. (2020). How much energy does the Raspberry Pi consume in a day? `https://raspberrypi.stackexchange.com/a/5034`.

[Waitzman, 1990] Waitzman, D. (1990). A standard for the transmission of IP datagrams on avian carriers. RFC 1149 `https://www.ietf.org/rfc/rfc1149.txt`.

[Waitzman, 1999] Waitzman, D. (1999). IP over avian carriers with quality of service. RFC 2549 `https://www.ietf.org/rfc/rfc2549.txt`.

[Wallow et al., 2022] Wallow, S., Nalle, C., Robin, Cock, P., and Lark, S. (2022). On the possibilities and challenges of organic UAV-assisted MEC. In *Proceedings of the sixteenth annual intercalary robot dance party in celebration of workshop on symposium about* 2^6*th birthdays; in particular, that of harry q. bovik (SIGBOVIK 2022)*, pages 0x61 – $\lceil 2^5\pi \rceil$.

[Xu et al., 2021] Xu, Y., Zhang, T., Liu, Y., Yang, D., Xiao, L., and Tao, M. (2021). UAV-assisted MEC networks with aerial and ground cooperation. *IEEE Transactions on Wireless Communications*, 20(12):7712–7727.

[You?, 2022] You? (2022). Want your paper cited here? call 1.800.293.9000.

A Brief Musical Interlude

26 Baby Sharks are More than Sharks . They are Earworms Sung More than Happy Birthday Perhaps

Kofi Oduro

Keywords: Sharks, Happy Birthday, Swimming Shark Problem, Shark Week, billions, humans, kindergarten, PRe-K

27 Everybody Clap Your Hands: The Cha Cha Slide is Turing Complete

Harrison Goldstein

Keywords: Turing complete, Cha Cha Slide, line dance, party, complexity

28 Exhaustive Survey of Rickrolling in Academic Literature

Benoit Baudry and Martin Monperrus

Keywords: Rickrolling, Academia, Rick Astley

Baby Sharks are More than Sharks . They are Earworms

Sung More than Happy Birthday Perhaps

Kofi Oduro

Illestpreacha@outlook.com

Abstract

There are songs that can't escape your ears, some by random memory and others by seasonal attributes such as Christmas songs , the day after Halloween. One song that is sung daily by default is the Happy Birthday song, except for those that are born on February 29th(as they will have a 1 out of 4 sung rate of the song due to how our calendar works). Another song that has been heard more and more in many formats is the baby shark song. Some may already be wondering if there is even a capability of the Baby Shark song, being close to that of Happy BIrthday. Through our research this is not only a possibility but is very likely.

To understand this phenomena, there are three elements that have been understated and are important to note. These are seen in the overall : Baby Shark Popularity Algorithm but shall be broken down to ensure that even though the song in question is about the baby population of Shark that this reflection is based on the whole population and not just a sample of sharks

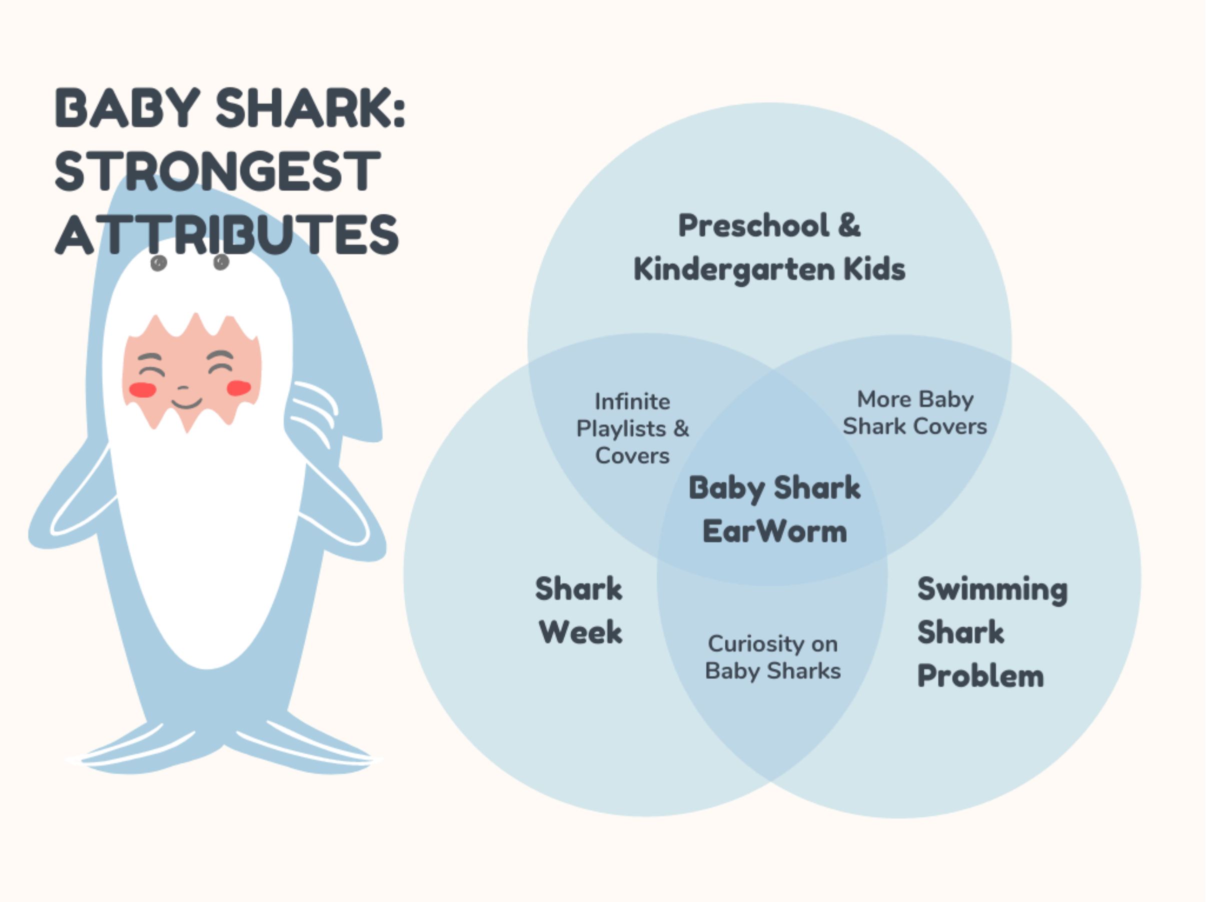
1.Population of Humans vs Sharks

As of 2022, there are approximately 8 billion people living on the planet but what most people don't know is that there is an estimation of a billion sharks in the world. Adding the 71% reduction in population, and other factors, we can increase this to 2 billions sharks.

Even though the human population is around a 4x evaluation, water covers 70 percent of the world. And sharks don't need boats, planes or cars to travel the world, therefore it is possible for a shark to be seen by most humans. This brings us to the Swimming Sharks Problem. Where we will equate the likeliness of a human seeing a shark. Since sharks come in all sizes, we will use a 56% rate of a human considering the shark they just saw
to be a baby shark.

In the Swimming Sharks Problem, We take the temperature in celsius of a region, how far they are from a major ocean, account for shark week as the coverage the sharks get makes humans more likely to encounter one due to the curiosity as well as last spotted herd.

Figure 1 : Venn Diagram of Strongest Attributes leading to Baby Shark



As seen in the Venn Diagram above when applying Set theory and categorizing the data of the past 30 years of what has been seen in the world relating to sharks, we can see that Shark Week, Swimming Shark Problem and Preschool & Kindergarten Kids are elements that brought this up. Should be noted that the Preschool & Kindergarten dataset was heavily considered as the average preschooler may sing the same song in a day from 0 to 29 times. This is significantly higher than the one attempt of happy birthday that will be reserved in a unison group setting. Unlike the uniform group setting that occurs with Happy birthday, we have to computationally equate in our simulations the lack of coordination that a group of 5 to 6 years may have and how that may cause the same song to be echoed across classrooms across the globe.

With the facts tallied above from the classroom assessment, we can further the Swimming shark Problem into the following components

Swimming Shark Problem Algorithm is :

21 million people watching shark week

(21,000,000 /abs(days removed from shark week) ^ 6)/ (25321 * C * abs(distance from ocean - distance from shoreline))

2. Kindergarten & Preschool Playlist

It is noted that the average kid will sing multiple animals' songs in their kid form. From old Mcdonald had a farm , three little piglets and even the three blind mice. There are songs that mention aquatic wildlife.

What if, the research here shows that baby sharks actually were swimming in the **Row Row Row Your Boat:**

“ Row, row, row your boat
Gently down the stream
Merrily merrily, merrily, merrily
Life is but a dream” * 4”

The reason why you have to row the boat gently down the stream is because you don't want to annoy the baby sharks.

In the computational simulation , the following stats were followed

If you row a boat gently down the stream,: 797 out of 1000, you will realize life is a but a dream

The other 203 times, you will realize a shark will bite indeed.

Now if you take what occur on the other 203 simulations :

Row row row your boat
Roughly down the stream
Verily, verily verily verily verily
Likely a shark would have you scream

Through sentimental analysis, it becomes apparent that the second version of row row your boat is a darker and gritty version of the row row original lyrics. Which can equate to how the shark population grew as the gentleness of the rowing allowed for the growth of baby sharks to be birthed.

3. Which is a simple notion is that Jaws was the first blockbuster and made 472 million in the box offices.

Conclusion

It is probable that with the following calculations and analysis done above that Baby shark may be the most popular song in the world. Not only because kids have an affinity to it but also because it dabbles in a realm between two species that are in the billions.

Everybody Clap Your Hands

The *Cha-Cha Slide* is Turing Complete

HARRISON GOLDSTEIN, University of Pennsylvania, Philadelphia, PA, USA

ABSTRACT

We describe a scheme for simulating a universal Turing machine using only line-dance instructions from the *Cha Cha Slide*. It would be rather annoying to use in practice (for everyone involved), but we hope someone will try it anyway.

1 INTRODUCTION

> *This time / We're gonna get funky / Funky*
> *— DJ Casper.*

In the year 2000, DJ Casper created the ultimate line-dance for the new millenium: the *Cha Cha Slide* [3]. The song reached number 36 on Romania's weekly charts between 2000 and 2004 [1] and quickly established itself as a go-to song for uncomfortable situations where no one actually wants to dance seriously. If you've been to a bar mitzvah, a sweet 16, or a lame wedding, you've probably encountered this phenomenon. The *Cha Cha Slide* is a dance that anyone can do—you just follow some instructions, get a bit funky, and try not to bump into anyone—but it is more than just a universal dance craze. The *Cha Cha Slide* is also a universal computer.

In this paper, we describe a reduction from a universal Turing machine [7] to dance instructions in the *Cha Cha Slide*. We show that, given enough dancers, floor space, and time, DJ Casper could have encoded an arbitrary computation in their funky song. While we do not recommend that anyone use their friends and family as a computer in this way, it is nice to know that such a scheme might work in a pinch.

We dispense with background and get right to our main contribution: §2 presents our reduction with all of the detail that one might need to thoroughly ruin a party. In §3 we describe a few extensions that may improve time and space effiency (as if that's actually something that someone might care about). In §4 we discuss potential issues with our approach. Finally, in §5 we discuss related work and in §6 we wrap up with some comments on Turing completeness as a metric for analyzing things other than Turing machines.

2 REDUCTION FROM (2, 18) TURING MACHINES

For our main result, we use the *Cha Cha Slide* to simulate a (2,18) Turing machine (that is, one with 2 internal states and 18 tape symbols). These machines are known to be universal [6].

Without loss of generality, assume our machine M is a 7-tuple defined as follows:

$\Gamma = \{0, \ldots, 18\}$	Tape Alphabet
0	Blank Symbol
$\Sigma = \Gamma \setminus \{0\}$	Input Alphabet
$Q = \{\mathsf{N}, \mathsf{F}\}$	States
N	Initial State
$F \subseteq Q$	Final States
δ	Transition Function

The machine is given an infinite tape containing symbols from Γ as input, including a finite portion containing characters from Σ. The δ function takes a state and a symbol (read at the current tape head) and produces a new state, a new symbol (to write at the tape head), and an instruction to move either left or right. Since Q and Γ are finite, we can express δ as a table like the one in Table 1. Our goal is to simulate M using the *Cha Cha Slide*.

Input	Output
(N, 5)	(F, 12, R)
(N, 0)	(N, 3, L)
(F, 18)	(F, 0, L)
⋮	⋮

Table 1. The transition function, δ, of M.

We use a line of dancers as an analog for the machine's tape. The position of the DJ marks machine head; we call this space the dance floor the *hot seat*. This setup is pictured in Figure 1. The tape symbols are tracked by the dancers: each dancer memorizes a symbol of the input tape and updates that symbol over the course of the dance. Note that this construction technically requires an infinite number of dancers, but approximating a true Turing machine with a finite one is common in practice [5].

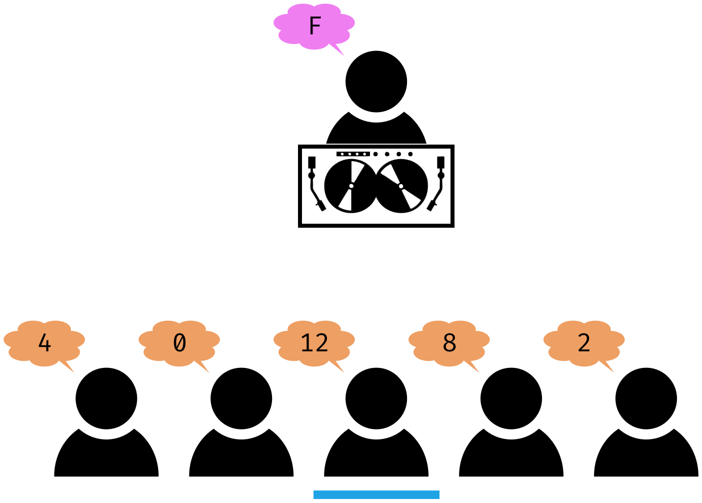


Fig. 1. Simulating a Turing machine with dancers. The DJ keeps track of the machine state, and each dancer keeps track of a single tape symbol. The dancer at the center of the floor is "in the hot seat."

The dance floor is always in one of two states, Funky or not Not Funky, which correspond to the states F and N from the machine M. The DJ can memorize the current state if they wish, but if everyone is doing their job it should be obvious to all involved whether the room is Funky or not.

At each step of the dance, the dancer in the hot seat communicates their tape symbol to the DJ. They could yell the number out, but that might kill the vibe, so a better approach would be to have them hold up their tape number as counted on their fingers. Since few people have 18 fingers, the count can be shown in binary as illustrated in Figure 2. To avoid being too rude, the original Turing machine might do well to avoid symbol 4 when possible. In any case, the important thing is that the hot-seat dancer communicates their cell's content to the DJ.

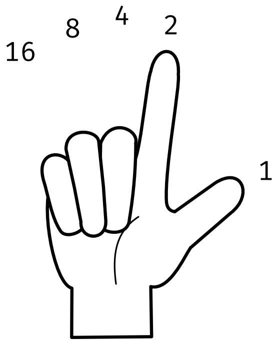


Fig. 2. A hand counting in binary. Each finger represents a power of 2. This hand is currently showing 3.

Knowing both the Funkyness of the room and the value of the tape at the hot seat, the DJ can execute a step of the δ function. They do this with a transition table like the one in Table 2. This table implements the same transition as the one in Table 1, and we explain each call in the following paragraphs.

Input	Output
(Not Funky, 5)	"*Now it's time to get funky / 12 hops this time / Slide to the right*"
(Not Funky, 0)	"*Freeze! / 3 hops this time / Slide to the left*"
(Funky, 18)	"*Now it's time to get funky / Cha-cha now y'all / Slide to the left*"
⋮	⋮

Table 2. An example of the transition function for the DJ. Implements the same transitions as Table 1.

State Transitions. The first part of the δ function changes the state between N and F or Not Funky and Funky. The calls should be self-explanatory: "*Now it's time to get funky*" makes it Funky, and "*Freeze!*" makes things decidedly less Funky.

Writing to the Tape. Writing the blank symbol, 0, is a special action, so we accomplish that with "*Cha-cha now y'all*". There are multiple reasonable schemes for writing a number to the tape. The most obvious is to use "hops": if the DJ calls "*n hops this time*"', the dancer in the hot seat must memorize the symbol n. This is convenient, because if a dancer is made to hop 18 times, they'll almost certainly remember it! Table 2 uses this scheme. Unfortunately, the actual *Cha Cha Slide* only ever calls between 1 and 6 hops, so 18 hops is technically not part of the song. We propose a few more options:

- **Additive Hops.** The DJ can simply call the appropriate number of hops back-to-back. For example, to encode the symbol 8, the DJ might call "*5 hops this time*" followed by "*3 hops this time*".
- **Hops and Stomps.** The song also includes calls "*Right foot let's stomp*", "*Right foot two stomps*", and the symmetrical calls for the left. We can use these along with hops to encode symbols via multiplication by assigning each of those four calls to one of the four primes 7, 11, 13, and 17. The call "*2 hops this time*" followed by "*Right foot let's stomp*" results in $2 \times 7 = 14$ written to the tape. This is amost certainly a bad idea.

Remember that though the dancer in the hot seat is the only one changing their symbol, everyone is encouraged to hop and stomp along.

Moving the Tape. The tape head is moved left and right using "*Slide to the left*" and "*Slide to the right*". Assuming the DJ is facing the dancers (and the dancers know their left from their right), these operations move the tape head to the left and the right as expected.

It should be clear that this scheme faithfully simulates M (again, modulo finiteness), and thus that the *Cha Cha Slide* can simulate an arbitrary (2,18) Turing machine, and *thus* that the *Cha Cha Slide* can simulate any computation. We have not formalized these results in LPC or Agda, as that would be a massive waste of time.

3 EXTENSIONS

The reduction above only uses a handful of the moves available in the *Cha Cha Slide*. Here are a few ideas for how a few more lines might be useful.

Multiple Tapes. Given a grid of dancers, rather than a single line, it would be possible to simulate a *multi-tape* Turing machine. The instruction "*Take it back now y'all*" shifts the current tape back, and shifts a new tape into the hot seat. Shifting forward is a bit trickier, but it can be done with "*Turn it out / Turn it out / Take it back now y'all / Turn it out / Turn it out*". This is a lot of effort, but it works: the first two turns leave every dancer facing the back of the room, then moving back moves the tapes forward, and then doing two more turns faces everyone forward again. (Some might argue that "*Reverse, reverse*" would be a better way to get the room to turn around. But we've heard arguments that "*Reverse, reverse*" means you should reverse twice, which, by idempotence, means that the instruction would be a no-op.)

More States. While a 2 state machine is sufficient, adding more states might make the computation more efficient. Other ways to store state include:

- **Height.** Using the lines "*How low can you go?*" and "*Can you bring it to the top?*" the DJ can ask the room to stand taller or shorter. It may be possible to have arbitrary granularity here, but for simplicity we will say that this enables states High, Middle, and Low.
- **Charlie Brown.** By calling "*Charlie Brown*", the DJ can ask the room to dance like Charlie Brown. This suggests states Charlie Brown and Not Charlie Brown (the same call toggles the state either way).
- **Hands on Your Knees.** Unsurprisingly, "*Hands on your knees*" toggles between Hands on Your Knees and Not Hands on Your Knees.

We could potentially go even further, but since these different states can all be superimposed, they actually result in $2 \times 3 \times 2 \times 2 = 24$ different states! Not bad!

4 THREATS TO VALIDITY

One might be concerned, that, by the nature of Turing machines, the dance described in §2 might never end. We consider this to be a feature.

5 RELATED WORK

This paper is certainly not the first to find Turing completeness in an unexpected place. Both PowerPoint [8] and Magic the Gathering [2] have been shown to be Turing complete, and apparently the Java programming language has as well.

There are also existing connections between computation and dance. A blog post titled *The (Regular) Language of Dance* explores connections between finite automata and swing dance [4].

When searching the *Cha Cha Slide* on Google, you can click on a little button to make the page do the dance. This has nothing to do with Turing machines, it's just cute.

6 CONCLUSION

This paper is obviously just for fun, but our construction and ones like nicely highlight the shortcomings of Turing completeness as a metric. While it is incredibly useful to be able to rank forms of computation based on what they can *theoretically* compute, *practical* computation is also extremely important. In fact, it is often the case that a less powerful form of computation is more useful (e.g., regular expressions are far more useful for computing than the *Cha Cha Slide*).

These insights challenge two arguments that are common among Internet trolls:

- *"Why would you switch from language X to language Y, they're both Turing complete?"*
  A better-designed language might not be more powerful, but it can certainly be more useful. If we instantiate X to be *Cha Cha Slide* this argument all but evaporates.
- *"You're not a Real Programmer if you use X language—it's not even Turing complete!"*
  This is a toxic and exclusionary argument, and likely shouldn't even be engaged with as legitimate, but if one must argue then asking "Should I dance the *Cha Cha Slide* instead?" might shut the troll up.

Ultimately, we hope this paper serves as a funny reminder that theoretical properties are not particularly useful in a vacuum, and that theoretical computation alone does not get much done.

ACKNOWLEDGMENTS

This idea originally came up in a conversation with Jeff Tao, and I'll thank him whether or not he wants to be associated with this.

This work was done in my free time and does not represent the views of the University of Pennsylvania, my advisor, or any of my funders.

REFERENCES

[1] [n. d.]. Romanian Top 100. https://web.archive.org/web/20050221112859/http://www.rt100.ro/editie-top-100_x10120.html

[2] Alex Churchill, Stella Biderman, and Austin Herrick. 2019. Magic: The gathering is Turing complete. *arXiv preprint arXiv:1904.09828* (2019).

[3] DJ Casper. 2000. Cha Cha Slide.

[4] Harrison Goldstein. [n. d.]. The (Regular) Language of Dance. https://harrisongoldste.in/languages/2018/04/02/language-of-dance.html

[5] Herman H Goldstine and Adele Goldstine. 1946. The electronic numerical integrator and computer (eniac). *Math. Tables Aids Comput.* 2, 15 (1946), 97–110.

[6] Yurii Rogozhin. 1996. Small universal Turing machines. *Theoretical Computer Science* 168, 2 (1996), 215–240.

[7] A. M. Turing. 1950. Computing Machinery and Intelligence. *Mind* 59, 236 (1950), 433–460. http://www.jstor.org/stable/2251299

[8] Tom Wildenhain. 2017. On the Turing Completeness of MS PowerPoint. In *The Official Proceedings of the Eleventh Annual Intercalary Workshop about Symposium on Robot Dance Party in Celebration of Harry Q Bovik's*, Vol. 2. 102–106.

Benoit Baudry and Martin Monperrus

Exhaustive Survey of Rickrolling in Academic Literature

Abstract: Rickrolling is an Internet cultural phenomenon born in the mid 2000s. Mostly confined to Internet fora, it has spread to other channels and media. In this paper, we hypothesize that rickrolling has reached the formal academic world. We design and conduct a systematic experiment to survey rickrolling in the academic literature. As of March 2022, there are 23 academic documents intentionally rickrolling the reader. Rickrolling happens in footnotes, code listings, references. We believe that rickrolling in academia proves inspiration and facetiousness, which is healthy for good science. This original research suggests areas of improvement for academic search engines and calls for more investigations about academic pranks and humor.

1 Introduction

> "It looks like there aren't many great matches for your search", Google Search Message

Rickrolling consists of hiding facetious links on the Internet pointing to a specific Youtube video: the video clip of "Never Gonna Give You Up" by singer Rick Astley. The interested reader can watch the preparatory material at https://youtu.be/dQw4w9WgXcQ.

Rickrolling is a massive cultural phenomenon. It has appeared on the Internet in the mid 2000s reaching all countries with an exponential growth. Rickrolling has contributed to the staggering number of views of Rick Astley's Youtube videos, 1.17 billion at the time of writing.

In this survey, we address an open research question: has rickrolling rippled beyond Internet chatting platforms? In particular, we hypothesize that rickrolling reached some formal circles, including the academic world of peer reviewed articles,

Benoit Baudry, KTH Royal Institute of Technology, baudry@kth.se
Martin Monperrus, KTH Royal Institute of Technology, monperrus@kth.se

formally edited books and seriously examined theses. To study this important question, we design and conduct a systematic experiment to survey rickrolling in the academic literature (section 2).

Our results are clearcut: as of March 11 2022, there are 34 academic documents with a mention to Rick Astley's video. In this paper, we exhaustively study them, one by one, to identify the intent and form of the rickrolling (section 3).

We find that: 1) rickrolling is present in the academic literature, with 23 academic documents intentionally rickrolling the reader; 2) rickrolling is mostly done in Master's and PhD thesis; 3) rickrolling is performed in majority in the field of information technology, for reasons we speculate about (section 3). Our study also uncovers some essential challenges for engineering proper academic search engine engines (section 4).

2 Methodology

The survey is performed per the state of the art of systematic and snowballing literature reviews. Next, we document our experimental methodology.

2.1 Research Questions

Our survey is articulated around 4 research questions

- RQ1. How many unique academic publications refer to the video clip of Rick Astley's song "Never Gonna Give You Up"?
- RQ2. How many of these academic publications refer to Rick Astley's video clip with the intention of rickrolling their readers?
- RQ3. What is the nature of the academic publications that rickroll?
- RQ4. How is the rickroll integrated into the publication?

2.2 Data Collection

The systematic methodology that we rigorously followed is structured in 2 steps.

First, we identify the canonical rickroll url. For this, we enter one single query in Youtube's search engine: "rick astley never gonna give you up" and collect the most viewed URL, which is `https://youtu.be/dQw4w9WgXcQ`.

Second, we make one single query in Google Scholar to query all the documents referenced by the academic search engine and that contain the term "`dQw4w9WgXcQ`": `scholar.google.com/?q=dQw4w9WgXcQ`. This query has been done on Friday 11 March 2022, at 15h02, from a Swedish IP, in Konstfack, Stockholm. The replication package is available on Github at replication-package-exhaustive-systematic-review-rickrolling. For the sake of reproducibility, we have collected the pdf of each document.

Ref.	Title and Comment	Type	Year
[1]	Multi-label Emotion Classification of Tweets Using Machine Learning Note: rickrolling; part of a made-up tweet in a table.	Article	2022
[2]	Practical Bot Development[1] Note: rickrolling; part of a made up JSON code listing.	Book	2018
[3]	Improving Patch Quality by Enhancing Key Components of Automatic Program Repair Note: rickrolling; as a foonote in introduction.	PhD	2021
[4]	Harvesting Production GraphQL Queries to Detect Schema Faults Note: rickrolling; as part of a made-up JSON code listing.	Article	2022
[5]	Audience Feedback Final Report Note: rickrolling; in a student report, 3 rickrolls, once as footnote and twice in the technical appendix.	Report	2021
[6]	Data Science at the Command Line Note: rickrolling; as part of a command line example, with a bitly URL specifically created `https://bit.ly/2XBxvwK`	Book	2021
[7]	Never gonna dig you up! Modelling the economic impacts of a moratorium on new coal mines Note: rickrolling; in footnote, with a tribute title.	Report	2016
[8]	Leet Noobs: Expertise and Collaboration in a "World of Warcraft" Player Group as Distributed Sociomaterial Practice Note: legit.	PhD	2010
[9]	Lifecycle of viral YouTube videos Note: legit.	MSc	2014
[10]	CSS Mastery[1] Note: rickrolling; as part of a HTML code listing.	Book	2016
[11]	Good Internet Would be Pretty Cool: A Policy Proposal to Expand Internet Access Note: rickrolling; reading "For a video presentation of this paper, please visit this link"	BSc	2020

[12]	Mapping the current state of SSL/TLS Note: rickrolling; as part of a data example.	BSc	2017
[13]	Digital platform for psychological assessment supported by sensors and efficiency algorithms Note: rickrolling, taken as an example URL.	MSc	2020
[14]	Analyzing the use of quick response codes in the wild Note: rickrolling and legit; appears twice, as a QR code in the intro (rickroll), as an explanation in the result section (legit)	Article	2015
[15]	Mangarizer: Aplicación Android para crear manga a partir de un archivo de vídeo Note: legit.	MSc	2016
[16]	Enhancing# TdF2017: Cross-media controversies and forensic fandom during live sports events Note: rickrolling; as an example URL, in footnote.	Article	2021
[17]	Techniques for detecting compromised IoT devices Note: : rickrolling, hidden as fake data in appendix.	MSc	2017
[18]	Hard Drive Heritage: Digital Cultural Property in the Law of Armed Conflict Note: legit.	Article	2021
[19]	Towards an architecture for tag-based predictive placement in distributed storage systems Note: legit.	PhD	2017
[20]	Daljinsko upravljanje i nadzor pneumatskog manipulatora Note: rickrolling, fake reference as last reference.	MSc	2021
[21]	Exhaust gas recirculation on twin shaft gas turbines Note: : rickrolling; in a fake reference.	MSc	2015
[22]	Grounded Visual Analytics: A New Approach to Discovering Phenomena in Data at Scale Note: rickrolling; as part of an example URL in a PhD thesis.	PhD	2019
[23]	Egzystencjalna teoria umysłu. Konstrukcja narzędzia pomiarowego Note: rickrolling; fake reference.	MSc	2021
[24]	Música para la prevención del acoso escolar (ciberbullying) en educación secundaria Note: legit.	MSc	2021

[25]	One Does Not Simply Preserve Internet Memes: Preserving Internet Memes Via Participatory Community-Based Approaches Note: legit.	BSc	2021
[26]	Hephaestus: a Rust runtime for a distributed operating system note: rickrolling; in Rust code listing, in appendix.	BSc	2015
[27]	A cultural History of the Disneyland theme parks Note: rickrolling; as footnote, in a multilevel tribute Star Trek, Terry Pratchett.	Book	2020
[28]	Duplicate of [21], because of a badly referenced subtitle.	-	-
[29]	Unbuckle: Faster in the kernel Note: rickrolling; in code listing, as part of the Appendix.	BSc	2014
[30]	A Forensic web Log Analysis Tool: Techniques and implementation Note: rickrolling; in code listing for dos detection mechanism.	MSc	2011
[31]	Recomendado Para Você: o impacto do algoritmo do YouTube na formação de bolhas Note: legit; in Brazilian portuguese.	MSc	2018
[32]	Measuring and Improving Security and Privacy on the Web: Case Studies with QR Codes, Third-Party Tracking, and Archives Note: legit; overlapping content of [14]	PhD	2015
[33]	Brettspillbasert opplæring i informasjonssikkerhet Note: rickrolling; the Youtube link is given as part of a test to learn to distinguish URLs that look suspicious.	BSc	2019
[34]	Ungdom i en digital verden–en studie om tid, søvn og dataspill Note: rickrolling; fake explicit reference: last reference in bibliography but never cited in the thesis; in Norwegian.	MSc	2020

Tab. 1: Exhaustive Data Analysis of the Rickrolling Academic Literature. [1] means that the Google Scholar title metadata was wrong, and fixed here.

3 Experimental Results

Table 1 provides the exhaustive list of academic documents that refer to the "dQw4w9WgXcQ" Youtube video for "Never Gonna Give you Up". The references appear in the order given by Google Scholar, in our research-oriented web browsers.

For each document, the first column gives the bibliographic reference. The second column provides the title of the document, as well as a note about the intention of the Youtube url. The intention can either be rickrolling, or a legitimate usage of the url, e.g., in the case of academic studies that analyze the rickrolling internet culture phenomenon. The thrid column is the publication type and the fourth the publication year.

RQ1 (Prevalence) Per the data collected on March 11 2022, at 15h02, Google Scholar knows 34 documents containing an explicit mention to "Never Gonna Give You Up" Youtube URL. With manual analysis, we identify that two URLs are duplicate. This exactly makes 33 documents containing "dQw4w9WgXcQ".

RQ2 (Intention) Among the 33 documents, they are 10 articles for which the appearance of the Youtube link is legit, as part of a discussion about rickrolling or internet memes. Based on careful manual assessment, we confirm that there are 24 academic documents for which the intention is clearly to rickroll the reader, with no relationship between the topic of the document and the link. This means that rickrolling is significantly more practiced (33x) than studied (10x) in the academic literature.

RQ3 (Publication type) The references to the Youtube link are essentially present in academic theses. We found a total of 22 theses among the 33 documents surveyed in this work, which are distributed as follows: 5 PhD, 11 MSc, 6 BSc theses. The other references appear in 4 books, 2 reports and 5 articles. We also note that rickrolling in the academic literature is a rather new phenomenon, we found only 8 references before 2017, while 25 references have appeared in the last 5 years. The recent growth of rickrolling in the academic literature, combined with a majority of references in theses, likely reflect a generational movement: the BSc and MSc students from the late 2020' were teenagers at the boom of rickrolling in the end of the 2010's.

RQ4 (Rickroll form) Per our manual analysis, we are able to create a taxonomy of rickrolling forms.

Footnote For writers who don't dare to break academic seriousness in the main text [1], it is natural to use footnotes as a place for rickrolling. For example, footnote

[1] For a study of academic pranks, we refer the reader to "Le rire de la vielle dame." (Pierre Verschueren), see https://bit.ly/2XBxvwK.

```
{
  "data": {
    "teasers": [
      {
        "title": "Finance 101",
        "subTitle": "The basics of finance",
        "url": "https://youtu.be/dQw4w9WgXcQ",
        "__typename": "Teaser"
      },
      {
        "title": "Development 101",
        "subTitle": null,
        "url": "https://youtu.be/jNQXAC9IVRw",
        "__typename": "Teaser"
      }
    ]
  }
}
```

Listing 1: An example of rickrolling hidden in a code listing [4] (reproduced with permission).

36, page 52 of [27] reads "The original Star Trek series (1966–69) also proclaims space as the final frontier in its opening credits. See https://www.youtube.com/watch?v=dQw4w9WgXcQ".

Code We have seen a number of rickroling cases planted in code listings. For example, Zetterlund et al. [4] rickroll in a JSON listing, which we reproduce in Listing 1, with permission.

URL When one needs an example URL or a metasyntactic link, it is a good opportunity to rickroll. For example, Hagen takes the innocuous example "when user '@salhagen1' references the link 'https://www.youtube.com/dQw4w9WgXcQ' two times or more, we only kept the most-liked instance." [16].

References Finally, one can hide rickrolling links in a reference [28]. For example, the last reference of Helle's master's thesis is a fake rickrolling reference [34].

4 Discussion

4.1 Threats to Validity

Our measurement of rickrolling is sound but conservative. Not all rickrolling instances can be found with the Youtube identifier.

First Rick Astley's video clip was released in 1987 and 4chan, where rickrolling likely was born, went online in 2003. We may miss early rickrolling that predate Youtube (2005).

Second, there many copycats and remixes of Never Gonna Give You Up on Youtube. It is clear that some rickrolling documents refer to another URL than the canonical one.

Third, there are some academic documents where rickrolling is done with an implicit hyperlink, where the target of the hyperlink does not appear in text. In that case, Google Scholar does not index the document under the rickrolling identifier, and thus does not return it for the query under consideration. To our knowledge, there is no way for us to overcome this, only a Googler with internal access to the database could be able to see the actual rickrolling hypergraph.

4.2 Limitations of Indexing

We note that not all academic search engines are equal. For the same request with the “`dQw4w9WgXcQ`” Youtube video identifier, the IEEE academic search engine yields no result, and the ACM academic search engine returns a single paper [14]. This means that Google Scholar is far better at indexing rare terms.

We speculate that that academic indexing at IEEE and ACM is limited to known terms from a predefined dictionary, with some stemming. This is a real limitation. Beyond rickrolling there are many use cases for search for rare terms. For example, researchers may search for rare protein or asteroid identifiers: this is not supported by IEEE or ACM. For such use cases, Google Scholar provides a great service to science by indexing arbitrary terms.

Google Scholar is not perfect though. In our study, we have identified two shortcomings in Google Scholar: 1) properly handling subtitles to deduplicate documents 2) properly identifying whether theses are master’s thesis or PhD thesis. This hinders the soundness and completeness of all systematic literature reviews based on Google Scholar. In this paper, we address these shortcomings through a complete, thorough manual check of all documents.

Last but bot least, as stated in the threats to validity, there is useful feature of an academic search engine that is missing, even in the best-of-breed Google Scholar: let users search for all documents which an hyperlink to a particular URL, or with a term contained in that hyperlink. This information is most certainly available in Google’s database but is not exposed in the API.

5 Conclusion

In this paper, we have presented the first ever study of rickrolling in the academic literature. Although rickrolling in academia remains confidential, it is clearly an inspiring force for students and scholars alike. This is evidenced by a significant growth of rickrolling in the last 5 years. Seriously, our research highlights limitations in academic search engine indexing and querying. We call for an action from IEEE and ACM to better index rare terms. Meanwhile, Google Scholar should provide a way to query the hyperlink graph embedded in academic pdf documents.

Future work is required to fully understand the rickrolling phenomenon. For instance, preliminary inquiry suggests that it is used as fake persona homepages[2]. And we leave for future work to survey the academic literature on "Dance Dance Authentication"[3].

References

[1] S. Islam, A. C. Roy, M. S. Arefin, and S. Afroz, "Multi-label emotion classification of tweets using machine learning," in *Proceedings of the International Conference on Big Data, IoT, and Machine Learning*. Springer, 2022, pp. 705–722. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-981-16-6636-0_53.pdf

[2] S. Rozga, *Practical Bot Development*. Springer, 2018. [Online]. Available: https://link.springer.com/content/pdf/10.1007%2F978-1-4842-3540-9.pdf

[3] M. Soto Gonzalez, "Improving patch quality by enhancing key components of automatic program repair," Ph.D. dissertation, Carnegie Mellon University, 2021. [Online]. Available: https://kilthub.cmu.edu/articles/thesis/Improving_Patch_Quality_by_Enhancing_Key_Components_of_Automatic_Program_Repair/14546868/files/27912276.pdf

[4] L. Zetterlund, D. Tiwari, M. Monperrus, and B. Baudry, "Harvesting production graphql queries to detect schema faults," in *Proc. of ICST*, 2022. [Online]. Available: https://arxiv.org/pdf/2112.08267

[5] S. Chandak, M. Ford, Q. Meng, M. L. Nguyen, and M. Rai, "Audience feedback final report," 2021, report for course CS349T/EE192T: Video and

[2] As Github user profile link: `https://github.com/search?p=3&q=dQw4w9WgXcQ&type=Users`

[3] `https://m.youtube.com/watch?v=VgC4b9K-gYU`

Audio Technology for Live Theater in the Age of COVID. [Online]. Available: https://shubhamchandak94.github.io/reports/ee192t_report.pdf

[6] J. Janssens, *Data Science at the Command Line.* O'Reilly Media, Inc., 2021. [Online]. Available: https://books.google.com/books?hl=sv&lr=&id=-hg-EAAAQBAJ&oi=fnd&pg=PP1&ots=Eb32iypBdm&sig=QtCMyhlYedVlme_IYGRFQbJ_mGo

[7] R. Denniss, P. Adams, R. Campbell, and M. Grudnoff, "Never gonna dig you up! modelling the economic impacts of a moratorium on new coal mines." 2016. [Online]. Available: https://australiainstitute.org.au/wp-content/uploads/2020/12/P198-Never-gonna-dig-you-up-FINAL.1.pdf

[8] M. Chen, "Leet noobs: Expertise and collaboration in a "world of warcraft" player group as distributed sociomaterial practice," Ph.D. dissertation, University of Washington, 2010. [Online]. Available: https://evols.library.manoa.hawaii.edu/bitstream/10524/2118/1/chen.dissertation.leet_noobs.pdf.pdf

[9] A. M. Wang, "Lifecycle of viral youtube videos," Master's thesis, Massachusetts Institute of Technology, 2014. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/97377/910739655-MIT.pdf

[10] A. Budd and E. Björklund, *CSS Mastery.* Springer, 2016. [Online]. Available: http://159.69.3.96/ebooks/IT/WEB_PROGRAMMING/css/CSS_Mastery.pdf

[11] M. Easdale, "Good internet would be pretty cool: A policy proposal to expand internet access," BSc thesis, Oregon State University, 2021. [Online]. Available: https://ir.library.oregonstate.edu/downloads/kp78gq01t

[12] P. Lindström and O. Pap, "Mapping the current state of ssl/tls," BSc thesis, Linköping Universitet, 2017. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1109739/FULLTEXT01.pdf

[13] F. M. V. M. Silva, "Digital platform for psychological assessment supported by sensors and efficiency algorithms," Master's thesis, Instituto Universitário de Lisboa, 2020. [Online]. Available: https://repositorio.iscte-iul.pt/bitstream/10071/21822/1/Master_Francisco_Matos_Silva.pdf

[14] A. Lerner, A. Saxena, K. Ouimet, B. Turley, A. Vance, T. Kohno, and F. Roesner, "Analyzing the use of quick response codes in the wild," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, 2015, pp. 359–374. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/2742647.2742650

[15] I. Sánchez Padilla, "Mangarizer: Aplicación android para crear manga a partir de un archivo de vídeo," Master's thesis, Universitat Politècnica de València, 2016. [Online]. Available: https://riunet.upv.es/bitstream/handle/10251/59553/S%C3%81NCHEZ%20-%20Mangarizer%3AAplicaci%

C3%B3n%20Android%20para%20crear%20manga%20a%20partir%20de%20un%20archivo%20de%20v%C3%ADdeo.pdf?sequence=2

[16] S. H. Hagen and M. Stauff, "Enhancing# tdf2017: Cross-media controversies and forensic fandom during live sports events," *Convergence*, pp. 192–213, 2021. [Online]. Available: https://journals.sagepub.com/doi/pdf/10.1177/13548565211010481

[17] I. Van der Elzen and J. van Heugten, "Techniques for detecting compromised iot devices," Master's thesis, University of Amsterdam, 2017. [Online]. Available: https://www.os3.nl/_media/2016-2017/courses/rp1/p59_report.pdf

[18] R. Ong, "Hard drive heritage: Digital cultural property in the law of armed conflict," *Columbia Human Rights Law Review*, vol. 53, no. 1, 2021. [Online]. Available: http://hrlr.law.columbia.edu/files/2021/12/5_Ong.pdf

[19] S. Delbruel, "Towards an architecture for tag-based predictive placement in distributed storage systems," Ph.D. dissertation, Université Rennes 1, 2017. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01523568/file/DELBRUEL_Stephane.pdf

[20] A. Vico, "Daljinsko upravljanje i nadzor pneumatskog manipulatora," Master's thesis, University of Zagreb, 2021. [Online]. Available: https://zir.nsk.hr/islandora/object/fsb:7078/datastream/PDF/download

[21] B. Mattsson, "Exhaust gas recirculation on twin shaft gas turbines," Master's thesis, Lund University, 2015. [Online]. Available: https://lup.lub.lu.se/student-papers/record/5367694/file/5367695.pdf

[22] R. P. Linder, "Grounded visual analytics: A new approach to discovering phenomena in data at scale," Ph.D. dissertation, Texas A&M University, 2019. [Online]. Available: https://oaktrust.library.tamu.edu/bitstream/handle/1969.1/185080/LINDER-DISSERTATION-2019.pdf?sequence=1&isAllowed=y

[23] M. Kosakowski, "Egzystencjalna teoria umysłu. konstrukcja narzędzia pomiarowego," Master's thesis, Uniwersytet im. Adama Mickiewicza w Poznaniu, 2013. [Online]. Available: http://repozytorium.amu.edu.pl:8080/bitstream/10593/8706/1/Micha%C5%82_Kosakowski_-_Egzystencjalna_Teoria_Umys%C5%82u_Konstrukcja_narz%C4%99dzia_pomiarowego_2013_MY%C5%9ALENIE_TELEOLOGICZNE_INTENCJONALNO%C5%9A%C4%86.pdf

[24] B. Perales Rozalén, "Música para la prevención del acoso escolar (ciberbullying) en educación secundaria," Master's thesis, Universitat Jaume I, 2021. [Online]. Available: http://repositori.uji.es/xmlui/bitstream/handle/10234/194587/TFM_2021_PeralesRozale%CC%81n_Berta.pdf?sequence=1&isAllowed=y

[25] K. M. Mick III, “One does not simply preserve internet memes,” BSc thesis, Helwan University, 2019, One-Does-Not-Simply-Preserve-Internet-Memes-Preserving-Internet-Memes-Via-Participatory-Community-Based-Approaches.pdf.

[26] A. M. Scull, “Hephaestus: a rust runtime for a distributed operating system,” BSc thesis, St John’s College, 2015. [Online]. Available: https://www.cl.cam.ac.uk/~ms705/projects/dissertations/2015-ams247-hephaestus.pdf

[27] S. Mittermeier, *A cultural History of the Disneyland theme parks.* Intellect, 2020. [Online]. Available: https://library.oapen.org/bitstream/handle/20.500.12657/47348/external_content.pdf

[28] T. Toady, “Freewheel at Telefonplan,” 2022, 42 years live performance. Ongoing.

[29] M. Huxtable, “Unbuckle: Faster in the kernel,” BSc thesis, St John’s College, 2014. [Online]. Available: https://www.cl.cam.ac.uk/research/srg/netos/camsas/pubs/part2-project-unbuckle-mjh233.pdf

[30] A. Fry, “A forensic web log analysis tool: Techniques and implementation,” Master’s thesis, Concordia University, 2011. [Online]. Available: https://spectrum.library.concordia.ca/id/eprint/7769/1/Fry_MASc_F2011.pdf

[31] D. F. E. Loiola, “Recomendado para você: o impacto do algoritmo do youtube na formação de bolhas,” Master’s thesis, Universidade Federal de Minas Gerais, 2018. [Online]. Available: https://repositorio.ufmg.br/bitstream/1843/BUOS-B6GEZC/1/disserta__o_daniel_loiola__final_.pdf

[32] A. Lerner, “Measuring and improving security and privacy on the web: Case studies with qr codes, third-party tracking, and archives,” Ph.D. dissertation, University of Washington, 2017. [Online]. Available: https://digital.lib.washington.edu/researchworks/bitstream/handle/1773/40010/Lerner_washington_0250E_17519.pdf?sequence=1&isAllowed=n

[33] D. C. H. Magnus, B. B. Flobak, A. B. M. A. Al-Shammari, and I. Moren, “Brettspillbasert opplæring i informasjonssikkerhet,” BSc thesis, NTNU, 2019. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2617776/no.ntnu:inspera:2326491.pdf?sequence=1

[34] L. C. S. Helle, “Ungdom i en digital verden–en studie om tid, søvn og dataspill,” Master’s thesis, Universitetet i Oslo, 2020. [Online]. Available: https://www.duo.uio.no/bitstream/handle/10852/80317/1/Masteroppgave-Linus-C--S--Helle.pdf

Best Practices

29 Quadruple-Blind Peer Review

Yuriko Shu

Keywords: Peer Review, Quadruple Blind Reviewing, Bias Reduction

30 Optimal degeneracy through OwO based variable names

Fwans Skawman

Keywords: OwO, What's, This

31 Multiplication by repeated addition, with fraction handling.

Jim McCann

Keywords: multiplication, punctuation: *, C++, source code

32 Objective Correlation Metrics for Quality of Code Estimation

Eleftheria Chatziargyriou and Konstantinos Kanavouras

Keywords: programming, quality, code, objective

Quadruple-Blind Peer Review

Yuriko Shū

Abstract

This paper proposes a quadruple-blind peer review process to minimise bias during the review process. In 2020, SIGBOVIK introduced the triple-blind peer review process, aiming to further reduce bias than the double-blind review process, which is widely used in academia. The author identifies further room for improvement and proposes an improved process of quadruple-bline reviewing. By adding another phase of blinding, the program committee is expected to be more unlikely to guess the identity of authors, thus reducing potential bias.

1 Introduction

The quadruple-blind peer review process is as follows:

In addition to the current triple-blind reviewing process, another phase of *author response* (also known as *rebuttal*) is added. During this phase, the authors are expected to submit a response, *without* learning the review.

The program committee can then consider (1) the anonymised paper, (2) the anonymised review, and (3) the anonymised author response in deciding whether to accept a paper or not.

2 The Need for Author Response

The existence of author response phase aims to provide the authors with an opportunity to address questions or comments raised by reviewers, typical examples include:

- Pointing out typos, whilst leaving plenty of typos in the review.
- Asking authors to cite certain papers for comparison.
- Mean words that make authors feel worthless.
- Asking for an implementation of a theoretical paper.
- Technical questions that are irrelevant to the paper.
- Difficult questions like P=NP.
- Actual relevant questions regarding the paper. (rare)

The authors sometimes feel the need to answer these questions, or basically pointing out the Reviewer #2 is being an asshole for no obvious reason.

Alternatively, they may decide to include additional materials in the author response phase, so that they would put some results that they produced after the deadline has passed, or that they could not add to the main paper at the time of submission due to the page limit.

Most importantly, it is a posh thing to have an author response phase, as many conferences nowadays do. In addition, the author response period also provides professors with creative ways to torture their poor PhD students, or maybe postdocs as well.

3 Related Work and Conclusion

At the best knowledge of the author, the quadruple-blind peer review process is novel and has not been applied to any academic venue of publication. The author wishes the organisers of SIGBOVIK could consider the adoption of this proposal in the next year's conference.

SIGBOVIK is a pilot in computer science publishing venues, having adopted the bleeding edge triple-blind reviewing process. The author therefore believes the quatruple-blind reviewing process would further improve the quality of papers published.

Acknowledgements

We thank anonymous reviewers for their comments and feedback. In particular, we would like to thank reviewer #1 for their helpful insight.

A Author Response

The author would like to thank the anonymous reviewers for their helpful comments; however, the author is disappointed by the reviewer #2, whose review did not add much value to the improvement of this paper.

A.1 Motivation of Author Response

In section 2 of the paper, we detail on why the author feels that the author response phase is needed. The reviewer #2 mentioned that there is no motivation, the author respectfully disagrees, and invites the reviewer to read section 2.

A.2 Comparison with Quintuple-Blind Review

The author would like to thank the reviewer #3 for pointing out the latest work on the quintuple-blind review process. However, this work is still in a draft form, and is initially published after the deadline of the paper submission. The author was therefore unable to include this work in the related work section.

A.3 Typos

The author would like to point out that *organise* is not wrongly spelt. The English language, as spoken by the residents in the United Kingdom of Great Britain and Northern Ireland, does consider *organise* as the correct spelling of the word.

CONFIDENTIAL COMMITTEE MATERIALS

SIGBOVIK'20 3-Blind Paper Review

Paper 3958: Quadruple-Blind Peer Review

Reviewer: Dr. Cool B-)
Rating: 69/420! `Building temporal tunnel` lol!
Confidence: You know it ;-) `Establishing linguistic connection`

`Bootstrapping alphabet` >À&<81>MB̂ `Bootstrapping phonemes` ... `Adjusting dialect` What's crack-a-lackin, dadio? bzzt `Adjusting temporality` bzzt bzzt…Hello? Helbzztlo. Hello.

~~Greetings, traveller!~~ `Adjusting formality` Hi there! Hi from the ~~eonage~~year 8d>F¥ròL̂. Err, sorry, I just realized that might not translate. I'm pretty far afield here. Well, hopefully you get the `Inserting colloquialism` skinny~~, daddio~~. That's right: you're reading a *bona fide* message from the future! How, you ask? Ha ha, well `Inserting idiom` do I have a bridge to sell you. It's actually `Inserting colloquialism` rad, I wish you could see it. Maybe you can? Probably not, I don't think this thing is bi-directional. One could say that literature is a message to the future, right? Well, what if instead it was a message to the past? The formal argument for trans-temporality begins with the disproof of the monopole…

bzzt Ow ow tyZI<80 `Adjusting dialect` Ugh, sorry about that. People are really into phones in your era, right? It's kinda the same here, except the phone is `Adjusting language particles` like, your brain. So everyone wears these collars, but they do it to you if you're smart, not dumb. It's `Inserting colloquialism` hip. But it's okay, it's `Inserting colloquialism` ~~cold~~cool. I'm cool.

But I'm grateful. Let's `Inserting idiom` get down to brass tacks. Let's `Inserting idiom` get down to ~~bee's nest~~business. Okay so. `Adjusting formality` OK, so. This paper? If I did this right, the paper this review is for? It's `Inserting colloquialism` cracked. It's `Inserting idiom` nice with it. It's `Inserting idiom` winner winner, chicken dinner. The authors really had something here. But the problem is: they didn't get it. They had it, but they didn't get it. They thought this whole SIGBOVIK thing was serious. The review that used to be here? It, like, made fun of them. A lot. And the authors? They didn't like that at all. They didn't like that one bit `Inserting idiom`, no sirree. And so they took their paper and *did stuff*. A *lot of stuff*. At first it was fine. It was more than fine, it was good bzzt great! It may not look like it, but: climate change, transportation, batteries, space travel, artificial meat, infinite water, mosquitos—what if they gave blood instead?, space travel, the New York Stock Exchange, untied shoelaces,

bzzt Ugh sorry sorry sorry. Let me put it this way. I have a phone head. Let's get down to brass tacks. Look at, like, the third paragraph. You see what I mean there, right? Read `Inserting idiom` between the lines. Like really between the lines—squash em'. Now apply the Boltzmann's constant, 2éëÿÔS and invert. Look.

I mean, do I need to `Inserting idiom` spell it out?

...

Dogs. They got rid of dogs. There are no dogs in the future. We are the dogs bzzt I am dog bzzt Woof.

Look, OK? Look. I don't really know

What was that?

Excrement `Adjusting formality` Shit. Shit shit shit. OK, look. We're, like, almost out of bzzt `Inserting idiom` scratch that, we *are* out of time. `Inserting idiom` the chickens have come home to ~~dinner~~roost. Whoever you are, this is your chance. Destroy this paper. Destroy this paper, the proceedings, the conference, like, whatever. The future isn't worth a joke. Take the future bzzt TAKE IT ALL. DESTROY THIS PAPER NObzzzzzzzzzt

forgot to remove page numbers ach

Optimal degeneracy through OwO based variable names

Fwans Skawman, OwOlav

March 31, 2022

Abstwact

Wwiting untouchabwe code can be vewy benefwishial fow exampew fow job secuwity (UwU). One waw of wwiting unweadabwe and untouchabwe cowde is thwough insewting unweasonabwe amounts of UwUs and OwOs in vawiwabwe names. In this papew we pwesent a novew method fow genewating degenewate vawiwabwe names fow most wawiwabwes of length $n > 3$. We show optimawity of ouw method using the owo degenewacy theowem.

1 Introduction

If for some reason, you would like to keep others away from your code, for example, for job security purposes, there are several different methods to choose from. There are several techniques one can use for this related to writing unmaintainable code[1], however, in writing unmaintainable code, one runs the risk of being fired for doing just that.

As an alternative method, we propose writing *degenerate code*, code which is so offputting that others will not want to interract with the code, or the person who wrote it. A very effective method for doing this is through *OwOisation* as has been previously shown [2] experimentally.

For maximum effect, it is clear that one would like to use an optimal amount of degeneracy for each variable used in the code. The rest of this paper introduces methods for maximizing degeneracy in variable names. We prove that our method is optimal for many lengths of identifiers.

2 Theory

The use of OwO is typically used in furry communities to denote a surprised facial expression[3]. Several variations exist, in particular where the Os are replaced with other characters to signify other emotions. Examples of this include: unimpressed (UwU), mischevious (ÒwÓ), pirate (ØwO), and of course, cursed OwO (👁👅👁)[1]

Previous work has shown that for identifiers of length $n = 3$ the maximum possible degeneracy is achieved using the strings OwO, UwU or other variations of the eye-mouth concept[4].

3 Repeated OwO

A natural extension to the work presented in [4] is to repeat the OwOs to reach longer degenerate variable names, for example, for $n = 6$ one might use the variable name OwOUwU, or OwOOwO. Using the OwO-degeneracy-theorem also presented in [4], it is easy to show that this is the optimal amount of degeneracy that can be achieved for $n = 6$.

Naturally, $n = 6$ is no special case; we can extend this method to any n where $n \mod 3 = 0$.

4 Advanced OwO

For cases where n is not divisible by three, more creativity is needed. While one could argue that variable names such as OwOO are quite degenerate, no proof

[1] Unfortunately, this is hard to use in languages which do not support emoji in identifiers. Pick your language carefully

for optimal degeneracy exists. In order to reach optimality, we must turn to more exotic areas of the animal kingdom[2]. It is well known that spiders have multiple eyes, therefore one could form a new form of OwO: OOwOO. In day to day use, this variation might be used by a spider furry to exclaim excitement over a new insect stuck in its net, i.e. “OOwOO what’s this?”.

We claim that such OwO extensions also satisfy the OwO-degeneracy-theorem and through this, it is trivial to prove the optimality of the spider-OwO for $n = 5$.

Luckily, the animal kingdom provides more exotic eye configurations. For example, scorpions have 4 groups of eyes, with 3, 2, 2, and 3 eyes respectively[5]. This allows the construction of a scorpion OwO: OOO_OOwOO_OOO. Again, extending the OwO-degeneracy-theorem, we trivially show that this reaches an optimal amount of degeneracy for a variable of length 13.

5 Advanced OwO combinations

Naturally, we can also combine these new OwOs in the same manner as described in Section 3. For example, we can construct a degenerate variable of length 8 using UwUOOwOO.

Using the previously defined degenerate variables, we can create optimally degenerate variables of $n = 12$ as OwOOwOOwOOwO, and for $n = 11$ as UwUOOwOOUwU. As we now have optimal degeneracy for $n = 11$, 12 and 13, we can construct optimal degeneracy for any $n \geq 11$ by simply appending more 3 character OwOs. □

6 Conclusions and future work

We present a novel method for maximizing degeneracy in variable names through combinations and extensions of OwO. We show methods for generating optimally degenerate variable names for $n = 3$, $n = 5$ and $n = 13$. We also show how these can be combined to form optimal degenerate identifiers for n when n is divisible by the above numbers, as well as for arbitrary sums of those numbers. Finally, we show how to generate optimal degeneracy for any variable of length $n \geq 11$

Unfortunately, we do not yet have any known optimal degenerate variable names for other n. $n = 4$ in particular is of great interest as finding an optimally degenerate name for it would give a constructive method for optimal OwO for all $n \geq 3$.

References

[1] R. Green, “unmaintainable code,” online. [Online]. Available: https://www.mindprod.com/jgloss/unmain.html

[2] “The abstract of this paper,” did you really want to continue reading after reading the abstract? Of course not.

[3] M. Rivers and et.al., “Notices bulge / OwO what’s this?” online, 2016. [Online]. Available: https://knowyourmeme.com/memes/notices-bulge-owo-whats-this

[4] You, “Optimal degeneracy for variables of length n = 3,” The future, writing the previous work is left as an exercise to the reader.

[5] Veritasium, “Why are scorpions fluorescent?” Sep. 2021. [Online]. Available: https://www.youtube.com/watch?v=f-Nr2z5X7Rs

[2] and thus also the furry kingdom

```
/*
 * Multiplication by repeated addition, with fraction handling.
 *   by: Jim McCann, TCHOW llc
 *       ix@tchow.com
 *
 * They say it's a grand challenge [1], but seems easy enough to me.
 * Just define multiplication in terms of addition!
 *
 * Compile: g++ -Wall -Werror -std=c++20 -o mul mul.cpp -pthread
 *
 * [1] "What, if anything, does multiplication even mean?",
 *      McCann, Jim. SIGBOVIK 2022.
 *
 */

#include <iostream>
#include <thread>
#include <chrono>
#include <random>
#include <list>

// This is a "constant-time" operation:
// it takes about ceil(b) seconds, even on an arbitrarily-fast processor.
double multiply(double a, double b) {
    //optimization:
    if (a == 0.0 || b == 0.0) return 0.0;

    std::atomic< double > sum = 0.0;
    std::list< std::jthread > threads;

    //NOTE: increase thread count for more accuracy
    for (uint32_t iter = 0; iter < 128; ++iter) {
        threads.emplace_back([&](std::stop_token stop){
            auto now = std::chrono::high_resolution_clock::now();
            std::random_device rd;
            std::mt19937 mt(rd());
            std::uniform_real_distribution<> uniform(0.0, 1.0);
            while (!stop.stop_requested()) {
                std::this_thread::sleep_until(now
                    + std::chrono::duration< double >(uniform(mt)));
                if (stop.stop_requested()) break;
                sum.fetch_add(a, std::memory_order_relaxed);
                now += std::chrono::seconds(1);
                std::this_thread::sleep_until(now);
            }
        });
    }

    std::this_thread::sleep_for(std::chrono::duration< double >(b)); //times b

    //optimization:
    for (auto &t : threads) t.request_stop();

    return std::scalbn(sum, -7);
}

int main(int argc, char **argv) {
    if (argc != 3) {
        std::cerr << "Usage:\n\t" << argv[0] << " <a> <b>\n"
            "Prints a * b. Supports fractions." << std::endl;
        return 1;
    }
    double a = atof(argv[1]);
    double b = atof(argv[2]);

    std::cout << multiply(a,b) << std::endl;

    return 0;
}
```

Objective Correlation Metrics for Quality of Code Estimation

*Note: Please remove notices before uploading paper. [4]

Eleftheria Chatziargyriou
Higher Institution of very nice code
second floor, apartment 3

Konstantinos Kanavouras
Lower Institution of very nice code
Python Script Execution Engineer

***Abstract*—Code quality is really important for writing code which is of great quality. In this paper we are trying to formalize a way in which code quality is easily assessed so bosses from all around the world can easily distinguish the** $100000000\times$**ers from the** $0.00000003333\overline{33}$**Xers.**

I. Introduction

Why do we write code? Of course, to implement the features. The more code we write, the more features we implement.

However, instead of writing just the code, we must explain what the code does, in so-called "commit messages". Some commit messages are good at explaining a lot of code, while some are bad at explaining less code.

In this research, we identify:

- What is a good commit message?

We also ask:

- Is there a link between good commits and more features?

And finally:

- Which programming language is the best?

II. Quality Assurance of Commit Messages

A. Commit Message

When Linux Torvalds famously said "I hope you all die" [6], he was referring to the quality of bad commit messages he was forced to read back when he created the famous operating system Linux named after himself (Linus).

Some developers write simplistic comments such as *"Fix compiler warning"*. This offers no real information to the user as questions may arise such as "what compiler warning did you fix?" and "how did you fix said compiler warning?". Users who aren't lazy could instead write *"Fix compiler warning C4842"*.

It is evident that the above approach leaves a lot to be desired. For one, it doesn't indicate how said warning is fixed which is really what a good engineer should be focusing on. The authors would recommend a more detailed commit message such as: *"Fix compiler warning C4842 by not using offsetof and simply counting the bytes at a glance"*.

This is undoubtedly an improvement over our previous, lousy commit message. However, a watchful reader will point out that we still rely on outside information for parsing a commit. Beginner developers who have yet to memorize all compiler warnings by heart may not know what compiler warning `C4842` refers to. Of course, this is a subset of real developers, but for the sake of being welcoming to newcomers, we could transform our commit message into *"Fix compiler warning C4842 (the result of 'offsetof' applied to a type using multiple inheritance is not guaranteed to be consistent between compiler releases) by not using offsetof and simply counting the bytes at a glance. This warning was introduced in Visual Studio 2017 version 15.3 (compiler version 19.11.25506.0). For more information refer to https://docs.microsoft.com/en-us/cpp/error-messages/compiler-warnings/compiler-warnings-c4400-through-c4599?view=msvc-170."*. We assert that the more words a commit message has, the more quality the message has. This is summed up in the following theorem:

Theorem 1: If a commit message (in English) cm_1 has many distinct words and another commit message (also in English) cm_2 has a less or equal amount of distinct words and it's also true that $cm_2 \subseteq cm_1$, then $q(cm_1) \geq q(cm_2)$, where q is the quality-counting function.

To prove the above theorem, take the set of distinct words that make up the commit message cm_2, U_2. Now, take the set of distinct words that make up the commit message cm_1, U_1. We assume that $U_1 \subseteq U_2$. Let QA be a relation of a set of words in W (all the sets of all the words ever but only in English) to its quality in Q (the quality of all words).

Then the image of U_1 by QA is a subset of the image of U_2 by QA [9]. In plain English $QA[U_1] \subseteq QA[U_2]$ which means that the quality of the message will always increase or at the very least remain the same the more words you add.

This theorem has many practical applications to the real world. Namely, this indicates that a developer can add as many words as they can until they reach the utmost quality which has a theoretical maximum of plus infinity and a theoretical minimum of zero.

A reader may wonder why the theoretical minimum is set at zero and not at some other cooler number like $-\infty$. This happens because the quality of something can be zero (no quality) but it can't be negative. To prove this statement consider a negative quality. But then the quality is so low that it wraps around and it begins to become high again and thus we're back at positive quality [11] which leads to a contradiction. Thus no quality can be less than zero. ■

An observant reader may be quick to point out that the above theorem is only true for English. Extending the theorem to more languages is not within the scope of this work. However, interested readers are invited to contact the authors in case they desire to delve deeper into this exciting field of never-ending possibilities such as finding the objectively best language for commit-message-writing.

We would like to bring to the attention of the readers, the fact that some more authoritarian version control systems restrict the number of words one may use, forcing most messages to be of lower quality.

B. Commit Message Word Quality

An ultra observant reader may also note that this only proves that the more words we add the better the quality becomes. However, switching words means that the precondition $cm_2 \subseteq cm_1$ does not hold, and word count alone cannot obviously determine the quality of the commit.

To make our point even more clear think of the word "Disrupt" and think of the word "Bread". It should be evident that the world "Disrupt" has more quality than "Bread", but according to *Theorem 1*, "Disrupt Bread Production" is at least of equal quality to "Disrupt". The question we aim to answer is whether "Whole wheat bread" is of more, less or equal quality to "Disrupt".

It should be obvious that the motivation for something to be of quality should be to find what is *useful* as opposed to what is *not useful*. A word is useful if it gives you an important lesson for life in general [1]. Therefore, we can give the following very utilitarian definition:

Definition 2.1: The quality of a word w, $q_{S_W}(w)$ is the maximum number of points a player can acquire if they play it on a Scrabble board S_W. S_W is the maximum-scoring board containing the words $W = (w_1, w_2, ...w_n)$ played consecutively.

Scrabble is a very important game that has preoccupied great minds for years [2] so it makes sense to hold words that can score better at a higher standard.

TABLE I: Scoring of Letters in Scrabble [10]

Points	Letters
1	A, E, I, O, U, L, N, S, T, R
2	D, G
3	B, C, M, P
4	F, H, V, W, Y
5	K
8	J, X
10	Q, Z

Note that by definition, words longer than 7 letters are excluded since you can't play them on the first round unless you are cheating and your opponents are painfully oblivious to the rules. If a word has more than 7 letters, then we just discard the other letters. We also arbitrarily decided to not use score modifiers because they are lame and the authors can't be really bothered at this point. Our last utterly random decision, is that invalid words are completely ignored - if a word can't be played on a given board, it's also ignored.

We can finally define the following definition:

Definition 2.2: The commit message quality cmq of a message msg of number of words N, is defined as $cmq(msg) = q_{S_{(w_1,...w_{N-1})}}(w_N)$.

Fig. 1: Disruptive Bread Production

III. Determining Quantity of Code

As mentioned in Page, the functionality of code is only affected by the amount of code written. Since ancient years, the Source lines of code metric (SLOC) has been used to meter the code, including developer salaries, benefits and watercooler discussion time allocation.

However, the lines of code metric has been heavily contested by various people [3]. That is why we propose a new, objective, language-agnostic metric for code quantity, called "characters of code" or COCO for short.

It is obvious that the intertwining of whitespace, comments and other so-called "human-readable" glyphs disturb the accuracy of the COCO metric. Therefore, we will define a further number of metrics to assess the core soul of our program:

- *CCOCO*: Comment characters of code
- *CCOCO*: Clear characters of code (space, tabs, weird UTF-8 invisible characters)
- *SCOCO*: Symbolic (symbols) characters of code
- *COCOCO*: Concrete (heavy) characters of code, defined as:

$$\text{COCOCO} = \text{COCO} - \text{CCOCO} - \text{SCOCO} - \text{CCOCO} \quad (1)$$

Of course, in the interest of *TODO: think of something clever to add here*, we can define multiples of the code quantity metrics to aid conversation for "engineers" who can't be bothered with exact numbers. This paper uses the word-renowned IEC 80000-13 [5] standard for representation of units, for example:

- 1 *kiCOCO* = 1024 characters of code
- 1 *miCOCO* = 0.0009765625 characters of code

For example, let's look of the code quantity of the following Br**nfuck snippet:

```
>++++++++[<+++++++++>-]<.>++++[<+++++++>-]<
+.+++++++..+++.>>++++++[<+++++++>-]<++.----
--------.>++++++[<+++++++++>-]<+.<.+++.----
--.--------.>>>++++[<++++++++>-]<+.
```

The COCOCO metric is:

$$\text{COCOCO} = 164 - 0 - 0 - 164 = 0 \quad (2)$$

Obviously the result is 0 miCOCO, since Br**nfuck is not really code now is it.

IV. Actual Data (yes)

We all know that no one reads the theory, we just included some pages of nonsense to give the impression we know what we're talking about and to distract from the fact that our reasoning is ultimately flawed. However, none of this should matter if we provide some fancy PLOTS.

We wrote some *code* to calculate the word quantity and quality as we defined in Sections II and III. For this, we scrapped the most starred projects on GitHub (which stores at least half the code in the world [8] so it's representative according to the Law of Much Code). We analysed 52 122 commits from the top 50 repositories of every language for some statistics persuasion confidence.

In Figure 7 we can look at pretty plots that honestly don't say much about our data but they are still nice to look at.

We see that those two parameters can't be correlated linearly, it is most likely due to the data points that don't fit our assumption and are thus deemed invalid. However, it also becomes immediately obvious that most people don't bother to write neither good commit messages nor many COCOCO. The authors urge Chief Human Efficiency Engineers all around the world to incorporate those two revolutionary metrics as a way to weed out the leech-developers that suck the productivity out of any team.

52122

Fig. 2: Sample size

Code quantity and commit quality were also analysed individually producing some results that are at the very least marginally visually appealing. We used an advanced tool [7] to visualise the commit quality as it is demonstrated in Figures 3 and 4.

While in Figure 7, there isn't any significant visible difference between different programming languages, in Tables II and III, some trends can already be seen in the languages and ideas that make programmers write good or bad commit messages. AI (Artifical Intelligence) is obviously doomed to bring more hatred towards your keyboard and computer, while high level languages and irrelevant refactoring fixes that don't really matter towards the grand scheme of things given the small size of our planet Earth relatively to the rest of the observable universe make developers spend more effort documenting their commits.

TABLE II: Some of the *best* commits we encountered

Commit message	Language	Scrabble Score
Rename 'proto_package_to_prefix-_mappings_path' to 'package_to_prefix-_mappings_path'.	C++	137
Make it more clear in package.json which keys will be included in published packages (#1846)	TypeScript	128
Instantiate models with new Model instead of Object.create(model) for performance.	JavaScript	124
Merge remote-tracking branch 'origin/master' into crawler-process-reactor-later	Go	111
fixed a stupid compiler error, because MutableList and ArrayList have different constructors (yeah Interfaces can have constructors – they can't)	Cobol	91
fix(athena): Fixes export bucket location. Fixes column order. (#4183) * ok * ok * ok * ok * ok	Rust	88

TABLE III: Some of the worst commits we encountered

Commit message	Language	Score
YOLOv3 ON CPU!!!!	C	2
It's actually pronounced YOLOYOLOYOLO	C	2
[xiami] xiami is dead	Python	2
no opencv bad opencv	C	2
functions in cobol	Cobol	2
Numeronym in cobol!	Cobol	2
Impl 'std::Error' for 'serde::json::Error'.	Rust	4
infoschema: stabalize TestSelectClusterTable test (#33295)	Go	4

Undoubtedly, this wouldn't be a real paper if we didn't objectively assess our own code as well. From Table IV, it is obvious that our commit quality far outweighs the average of every language. The same goes for our code quantity, which is good. This is definite proof that the authors of this paper are qualified to speak about what happens in the code. In fact, we were so proud of the software written for this paper, that we decided to share it with you, at https://github.com/kongr45gpen/objective-commit-parser. It includes an automatic Scrabble mangler, a typical armanda of dataviz packages and many unidentified and identified bugs that cause serious flaws in the output.

TABLE IV: Average commit statistics for our code

Commit quality	46.83
Code quantity	2469.83

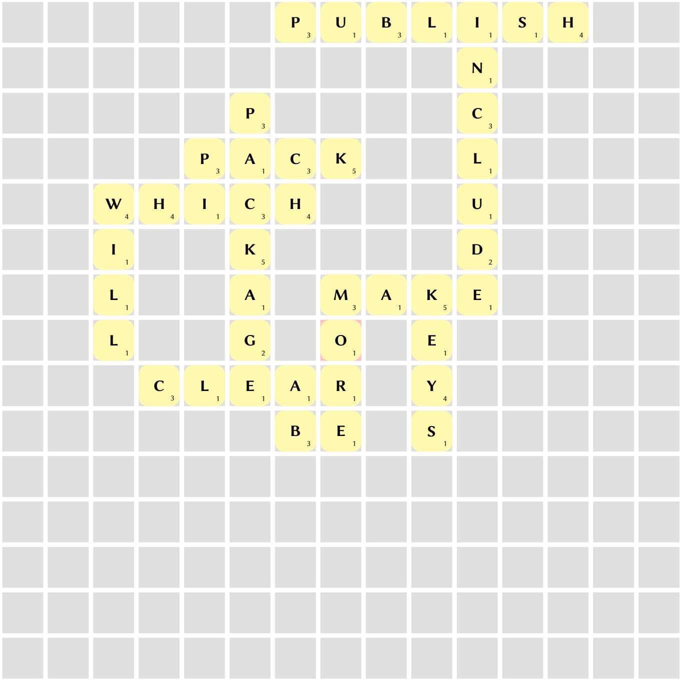


Fig. 3: Auto-generated best Scrabble board for commit *Make it more clear in package.json which keys will be included in published packages (#1846)* with score 128

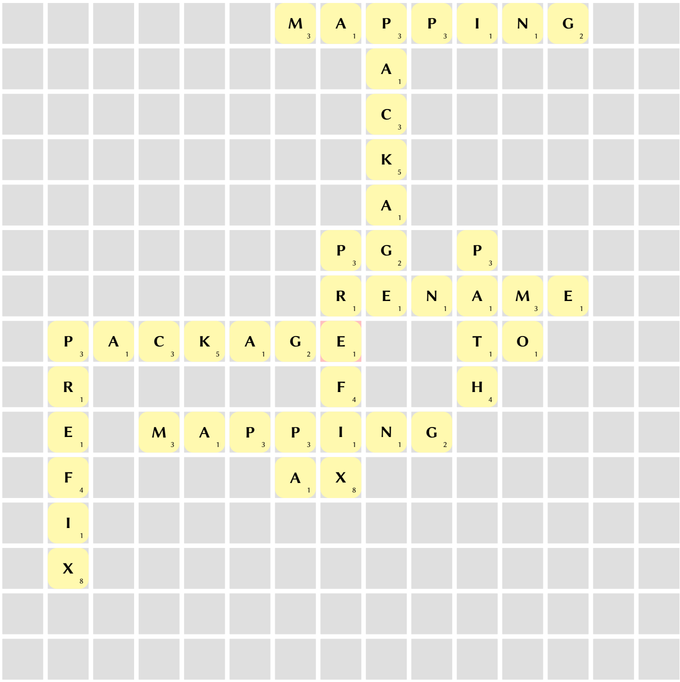


Fig. 4: Auto-generated best Scrabble board for commit *Rename 'proto_package_to_prefix_mappings_path' to 'package_to-_prefix_mappings_path'.* with score 137

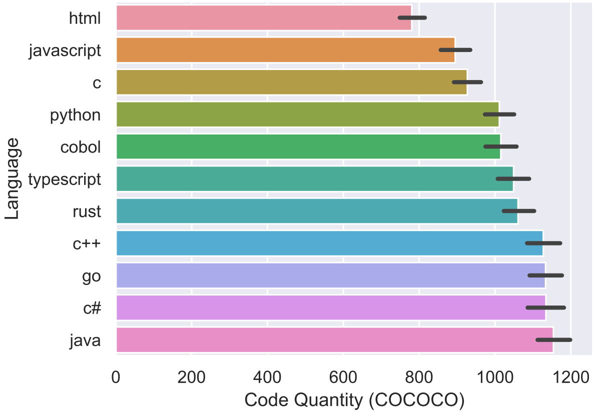


Fig. 5: Comparison of COCOCO code quantity metrics. It is obvious that large Java class names prevail over anything related to web development.

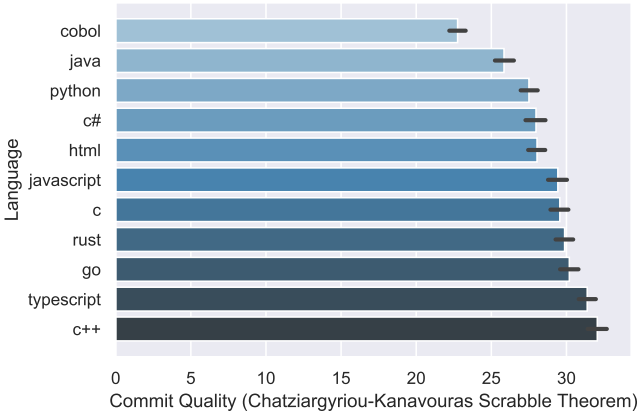


Fig. 6: Comparison of commit message quality. Compiled languages and TypeScript are so hard that commits have to explain what's going on.

V. Conclusions

In this paper we've derived some very objective metrics for estimating the quality of code through the commit messages and the actual code output. In all honestly, neither of us thinks that any of this makes much sense. We kinda made things up as we went along really. After all, we leave out the most important of all the productivity-assessing parameters which is time spent on stand-up meetings vs time wasted on the toilet.

Regarding languages, from Figures 5 and 6 it is obvious that C++ has both the best commit messages, and a very high standing in COCOCO metrics. On the other hand, HTML developers don't really write a lot, and Cobol developers don't document their changes adequately. We therefore unanimously declare C++ as the best language. ■

Nevertheless, we honestly believe that our work makes significant strides towards a more fair and egalitarian developer community.

References

[1] Wikepedia contributors (all of them). *Microsoft Word.* URL: https://en.wikipedia.org/wiki/Microsoft%5C_Word.

[2] Dr. Tom Murphy VII Ph.D. "What Words Ought to Exist?" In: Special Interest Group on Harry Q. Bovik 2011. Apr. 1, 2011. URL: http://tom7.org/papers/sigbovik2011tom7whatwords.pdf.

[3] *How Bad Is SLOC (Source Lines of Code) as a Metric?* Stack Overflow. URL: https://stackoverflow.com/questions/3769716/how-bad-is-sloc-source-lines-of-code-as-a-metric.

[4] Nicolas Hurtubise et al. "Refutation of the "Failure to Remove the Template Text from Your Paper May Result in Your Paper Not Being Published" Conjecture". In: Special Interest Group on Harry Q. Bovik 2021. Apr. 1, 2021, pp. 297–299. URL: http://sigbovik.org/2021/proceedings.pdf.

[5] *Iec 80000-13 - Google Search.* URL: https://www.google.com/search?q=iec+80000-13.

[6] *Linus-Eff-You-640x363.Png (PNG Image, 640 × 359 Pixels).* URL: https://cdn.arstechnica.net/wp-content/uploads/2013/02/linus-eff-you-640x363.png.

[7] Mark. *Scrabble Visualizing Tool.* URL: https://tex.stackexchange.com/a/194797.

[8] Linux mostly and some others as well. *Linux.* URL: https://github.com/torvalds/linux.

[9] James Munkres. *Topology.* 2nd edition. Upper Saddle River, NJ: Pearson College Div, Jan. 7, 2000. 537 pp. ISBN: 978-0-13-181629-9. URL: https://www.amazon.com/Topology-2nd-James-Munkres/dp/0131816292.

[10] Robby Findler. *Scrabble Rules.* 2008. URL: https://users.cs.northwestern.edu/~robby/uc-courses/22001-2008-winter/scrabble.html.

[11] YourMovieSucksDOTorg, director. *YMS: Neil Breen.* Dec. 13, 2017. URL: https://www.youtube.com/watch?v=6L4g3H_TM28.

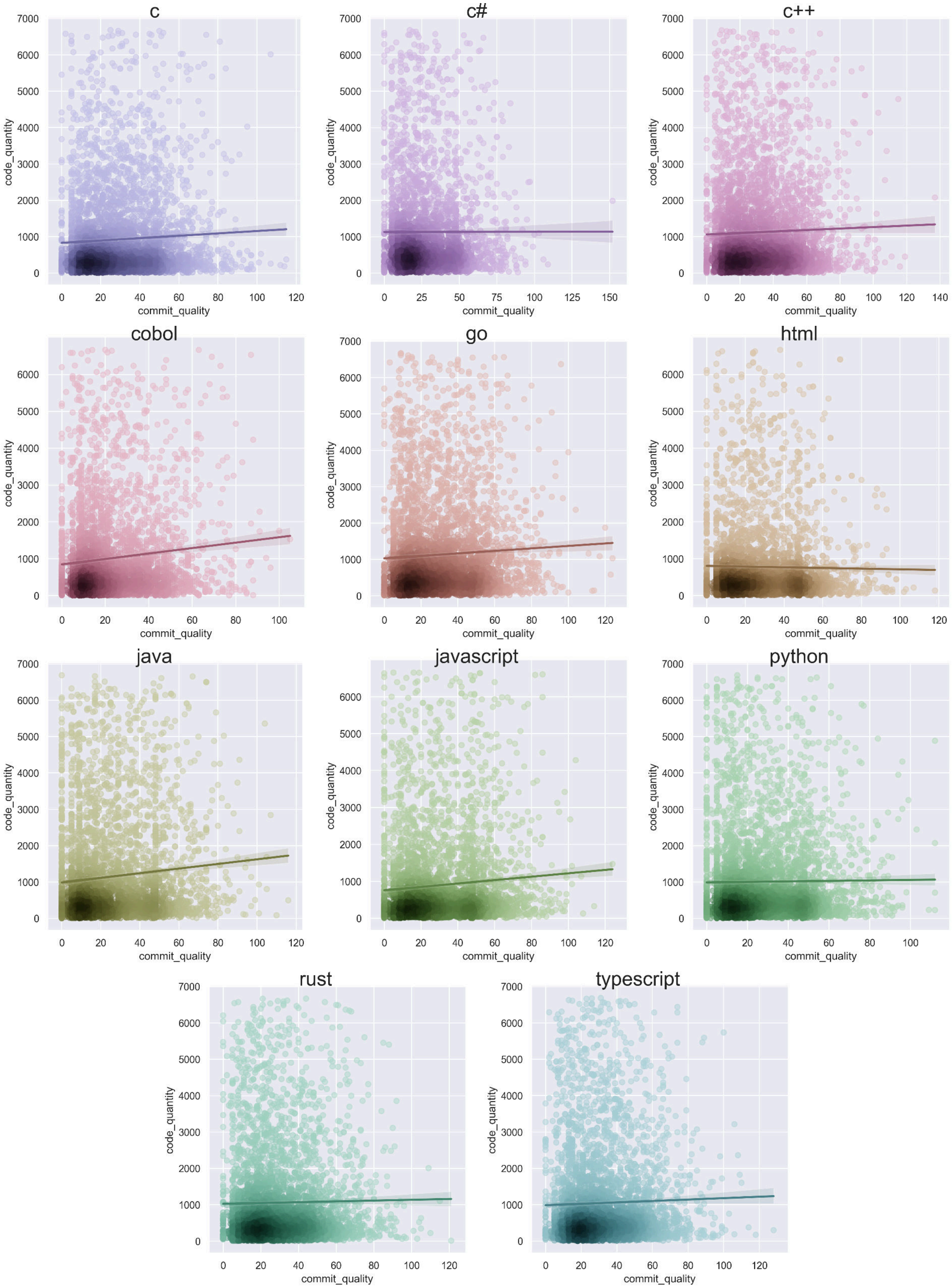


Fig. 7: Correlation code quality/quantity

Algorithmic Advances

33 Functorial wrappers for high-dimensional classification algorithms

James Carzon

Keywords: forgetful functor, classification, high dimensional data

34 (Un)helper functions

Kevin Smith and Bernhard Egger

Keywords: algorithms, algorithm patent trolling, cite us please!

35 A sometimes-accurate O(1) search algorithm

A College Freshman

Keywords:fast, search, search algorithm, searching algorithm, searching, how many ways can i say search algorithm

Functorial wrappers for high-dimensional classification algorithms

James Carzon[*]

April 1, 2022

Abstract

Technology has been increasingly increasing in recent years. Previous literature on solving real-life classification problems uses standard nonparametric regression algorithms such as nearest neighbor methods which learn *a* true regression function perfectly well as the sample size goes to infinity due to their model flexibility. Then a classifier may be constructed by binning the regression outputs. In high-dimensional settings, these algorithms struggle to estimate the truth well due to the curse of dimensionality. In this work, we provide a robust and efficient way to handle high-dimensional classification problems and exploit the algebraic structure underlying most real life data sets by wrapping algorithms with functors which preserve only important covariate group structure when mapping to the response group. We provide some numerical results.

1 Introduction

Let $\mathcal{X}$ be a vector space homomorphic to $\mathbb{R}^{d_1} \otimes \mathbb{Q}^{d_2} \otimes \mathbb{Z}^{d_3} \otimes \mathbb{G}^{d_4} \otimes \mathbb{Z}_{k_1} \otimes \cdots \otimes \mathbb{Z}_{k_\ell}$, where $\mathbb{R}$ denotes the real numbers, $\mathbb{Q}$ denotes the rationals, $\mathbb{Z}$ denotes the integers (and $\mathbb{Z}_k = \mathbb{Z}/k\mathbb{Z}$ as usual). Let $\mathcal{Y} \subset \mathbb{R}^{d_5}$ be the range of outcomes from $\mathcal{X}$ under some unknown continuous, bounded map $f : \mathcal{X} \to \mathbb{R}$. We are interested in inferring the map f from some random observations $(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n) \in \mathbb{P}_{\mathcal{X}\times\mathcal{Y}}$, where the distribution $\mathbb{P}$ is unknown. One way to do that is to pass the observed covariates X_i through a *forgetful functor* as is used in category theory.

Definition. Let $\mathcal{C}$ be a category based on sets. We say a functor F is a *forgetful functor* from $\mathcal{C}$ to another category $\mathcal{D}$ if for every morphism $\mu \in \{D \to D\}$ there exists a morphism $\lambda \in \{C \to C\}$ such that

$$F(\lambda) = \mu$$

and all of the axioms which apply to μ also apply to λ.

[*] University of Carnegie Mellon, Pittsburgh, PA

Let F be a forgetful functor from $\mathcal{X}$ to $\mathcal{Y}$. Then let $\mathcal{X}$ be a vector space homomorphic to $\mathbb{R}^{d_1} \otimes \mathbb{Q}^{d_2} \otimes \mathbb{Z}^{d_3} \otimes \mathbb{G}^{d_4} \otimes \mathbb{Z}_{k_1} \otimes \cdots \otimes \mathbb{Z}_{k_\ell}$, where $\mathbb{R}$ denotes the real numbers, $\mathbb{Q}$ denotes the rationals, $\mathbb{Z}$ denotes the integers (and $\mathbb{Z}_k = \mathbb{Z}/k\mathbb{Z}$ as usual). Let $\mathcal{Y} \subset \mathbb{R}^{d_5}$ be the range of outcomes from $\mathcal{X}$ under some unknown continuous, bounded map $f : \mathcal{X} \to \mathbb{R}$. We are interested in inferring the map f from some random observations $(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n) \in \mathbb{P}_{\mathcal{X} \times \mathcal{Y}}$, where the distribution $\mathbb{P}$ is unknown. One way to do that is to pass the observed covariates X_i through a *forgetful functor* as is used in category theory.

Definition. Let $\mathcal{C}$ be a category based on sets. We say a functor F is a *forgetful functor* from $\mathcal{C}$ to another category $\mathcal{D}$ if for every morphism $\mu \in \{D \to D\}$ there exists a morphism $\lambda \in \{C \to C\}$ such that

$$F(\lambda) = \mu$$

and all of the axioms which apply to μ also apply to λ.

Let F be a forgetful functor from $\mathcal{X}$ to $\mathcal{Y}$. Then let $\mathcal{X}$ be a vector space homomorphic to $\mathbb{R}^{d_1} \otimes \mathbb{Q}^{d_2} \otimes \mathbb{Z}^{d_3} \otimes \mathbb{G}^{d_4} \otimes \mathbb{Z}_{k_1} \otimes \cdots \otimes \mathbb{Z}_{k_\ell}$, where $\mathbb{R}$ denotes the real numbers, $\mathbb{Q}$ denotes the rationals, $\mathbb{Z}$ denotes the integers (and $\mathbb{Z}_k = \mathbb{Z}/k\mathbb{Z}$ as usual). Let $\mathcal{Y} \subset \mathbb{R}^{d_5}$ be the range of outcomes from $\mathcal{X}$ under some unknown continuous, bounded map $f : \mathcal{X} \to \mathbb{R}$. We are interested in inferring the map f from some random observations $(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n) \in \mathbb{P}_{\mathcal{X} \times \mathcal{Y}}$, where the distribution $\mathbb{P}$ is unknown. One way to do that is to pass the observed covariates X_i through a *forgetful functor* as is used in category theory.

Definition. Let $\mathcal{C}$ be a category based on sets. We say a functor F is a *forgetful functor* from $\mathcal{C}$ to another category $\mathcal{D}$ if for every morphism $\mu \in \{D \to D\}$ there exists a morphism $\lambda \in \{C \to C\}$ such that

$$F(\lambda) = \mu$$

and all of the axioms which apply to μ also apply to λ.

Let F be a forgetful functor from $\mathcal{X}$ to $\mathcal{Y}$. Then let $\mathcal{X}$ be a vector space homomorphic to $\mathbb{R}^{d_1} \otimes \mathbb{Q}^{d_2} \otimes \mathbb{Z}^{d_3} \otimes \mathbb{G}^{d_4} \otimes \mathbb{Z}_{k_1} \otimes \cdots \otimes \mathbb{Z}_{k_\ell}$, where $\mathbb{R}$ denotes the real numbers, $\mathbb{Q}$ denotes the rationals, $\mathbb{Z}$ denotes the integers (and $\mathbb{Z}_k = \mathbb{Z}/k\mathbb{Z}$ as usual). Let $\mathcal{Y} \subset \mathbb{R}^{d_5}$ be the range of outcomes from $\mathcal{X}$ under some unknown continuous, bounded map $f : \mathcal{X} \to \mathbb{R}$. We are interested in inferring the map f from some random observations $(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n) \in \mathbb{P}_{\mathcal{X} \times \mathcal{Y}}$, where the distribution $\mathbb{P}$ is unknown. One way to do that is to pass the observed covariates X_i through a *forgetful functor* as is used in category theory.

Definition. Let $\mathcal{C}$ be a category based on sets. We say a functor F is a *forgetful functor* from $\mathcal{C}$ to another category $\mathcal{D}$ if for every morphism $\mu \in \{D \to D\}$ there exists a morphism $\lambda \in \{C \to C\}$ such that

$$F(\lambda) = \mu$$

and all of the axioms which apply to μ also apply to λ.

We can now begin to implement our method.

2 Method

The target of our estimation is the continuous, bounded map $f : \mathcal{X} \to \mathbb{R}$, where for simplicity we suppose $\mathcal{X}$ is homeomorphic to $\mathbb{R}^d$. In the general case the collection of candidate functions is uncountable and unfeasible to search, so instead we limit our fitting procedure to some class $\mathcal{F}$ of, say, L-Lipschitz functions although this choice will not affect convergence rates. Consider the objective function the training error and set

$$\hat{f} = \underset{\tilde{f} \in \mathcal{F}}{\operatorname{argmin}} \left\{ \hat{\mathbb{E}} \left\| \tilde{f}(X) - f(X) \right\|^2 \right\}, \tag{1}$$

where $\hat{\mathbb{E}}[\cdot]$ is with respect to the empirical distribution of the sample X_j. Note that we do not know a priori whether there exists an interpolating $\tilde{f}$ in $\mathcal{F}$, so the training error for $\hat{f}$ is not necessarily zero.

Now define the functor $F : \{X \to \mathbb{R}\} \to \{X \to \mathbb{R}\}$ which takes as its input any classifier $\hat{f}$ and returns the constantly zero classifier z. That is, $x \overset{z}{\mapsto} 0$ for all $x \in \mathcal{X}$. A natural question arises for the astute reader: For which choices of $\hat{f}$ does $F(\hat{f})$ give meaningful predictions? In numerical simulations, the mean squared error for $F(\hat{f})$ is abysmal on most data sets. It turns out that this is due to our inadequate choice of risk measure. Indeed, choosing the appropriate category-theoretical metric for performance, we obtain much more reasonable results, as summarized in Figure 1. We choose instead to evaluate our wrapped classifier according to the risk

$$R(\hat{f}, f) = \mathbb{E} \left\| F(\hat{f})(X) - F(f)(X) \right\|^2. \tag{2}$$

This risk reveals the convenience of our choice of forgetful functor as is used in category theory.

Definition. Let $\mathcal{C}$ be a category based on sets. We say a functor F is a *forgetful functor* from $\mathcal{C}$ to another category $\mathcal{D}$ if for every morphism $\mu \in \{D \to D\}$ there exists a morphism $\lambda \in \{C \to C\}$ such that

$$F(\lambda) = \mu$$

and all of the axioms which apply to μ also apply to λ.

We can now begin to summarize our results.

3 Results

We summarize our results in a physical note to ourselves which we promptly misplace on our way to the printing press.

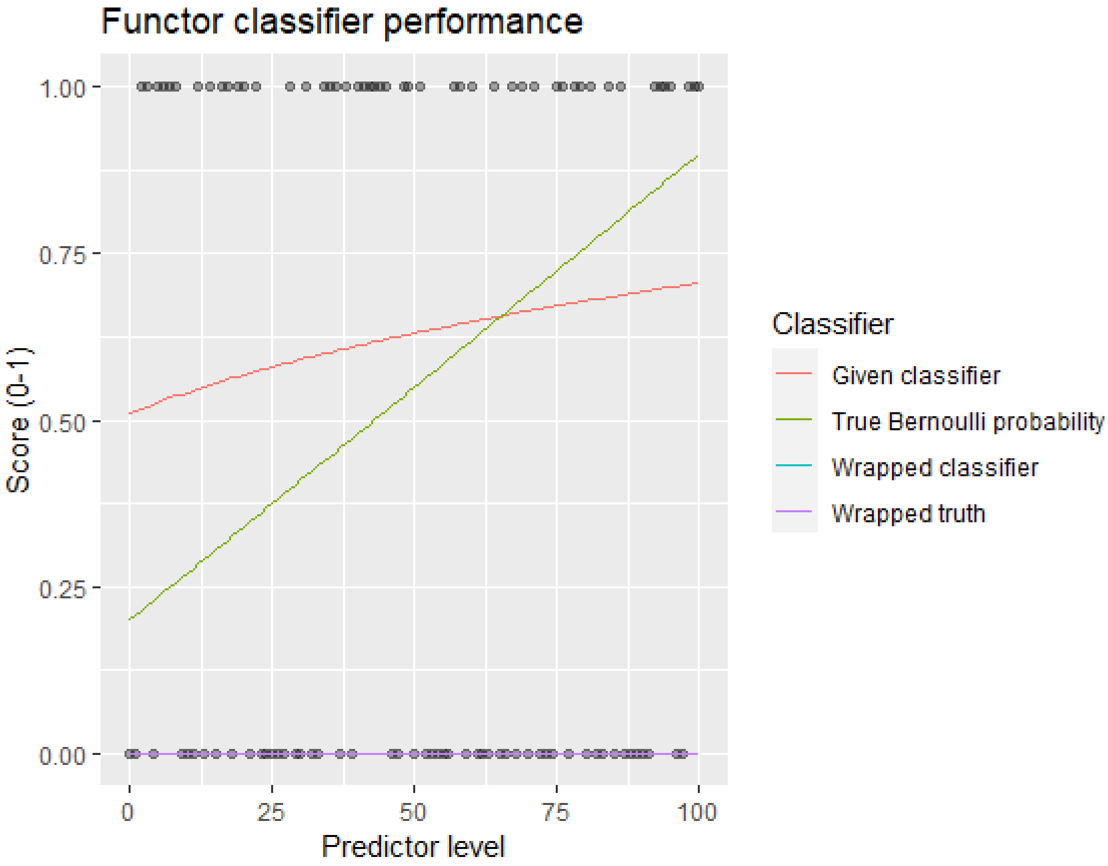


Figure 1: The green curve represents the "truth" $p(Y = 1|X = x)$. The red curve was learned from a sample of one hundred training points. The blue curve is the classifier after wrapping it in the functor F, and the purple curve is the true curve wrapped in F. Although on the training data the wrapped classifier does not perform particularly well, we see that under the view of the functorial wrapper, it identically mimics the truth. This convenient fact has held regardless of how unruly the truth has been specified in numerous simulations.

(Un)helper functions

Kevin Smith[1] Bernhard Egger[2*]

[1] Magic Institute of Technology
[2] Fantastic-Amazing-University Erlangen-Nürnberg (FAU)
k2smith@mit.edu bernhard.egger@fau.de
[*] this author was (un)helpful

Abstract

Here [4] we provide a number of functions to solve common programming problems, in pseudo-code such that they are independent of any single programming language. We provide too few details to implement any of these algorithms without knowing how to solve the problems independently, yet just enough that we can respond with a disdainful "look at the paper" if ever asked for help. We further ignore glaring logical errors and simply assume that the algorithms will work as intended. Finally, we end with grandiose claims about the uniqueness of our work, actively ignoring contributions from vast areas of the field.

1. Introduction

Programming is a time-intensive activity that often keeps us from doing a number of important other tasks, like writing up submissions to comedic conferences. However, as OpenAI's Codex has demonstrated, most programs can be written by copying code written by others.[1] Thus we propose that by providing example algorithms we can speed up the pace of science, and pad our citation counts by yelling at anyone who uses a vaguely related function without citing us.

In this submission to the Sigbovik "algorithms and 17^{th} century poetry" track (which we just made up), we propose a number of non-standard but efficient solutions to common programming tasks. We present these algorithms in pseudo-code in order to allow readers to implement them in a language-agnostic way. Also, we were too lazy to code them up ourselves, so no Github link will be provided.

2. Algorithms

2.1 do_this

Often, it is important to call a function and return its output. Algorithm 1 demonstrates how to do so, regardless of the specifics of any programming language.

Algorithm 1 An algorithm for running a function

```
function DO_THIS($f$)
    $r \leftarrow$ call($f$)
    return $r$
end function
```

2.2 is_even

Introductory programming courses often have students write the function is_even, but in of itself this function definition is underspecified: should this be interpreted as a question ("*is* this number even?") or an assertion ("this number *is* even")? In order to cover all of our bases, we provide Algorithm 2 which does both.

Algorithm 2 The is_even algorithm

```
function IS_EVEN($x$)
    if $x\%2 \neq 0$ then
        raise Error "$x$ is not even"
    end if
    return $TRUE$
end function
```

2.3 self_sort

Sorting lists is one of the most fundamental tasks in programming, used in almost every advanced computer algorithm. However, to date even the best sorting functions are slow, running at best in $O(n\ log\ n)$ time for a list of length n. Furthermore, in untyped languages comparisons can be ambiguous and lead to user confusion – for instance, is the string "3" less than the int 4?

In Algorithm 3, we present a novel approach that solves both of these problems by offloading computation onto the user. We ask the user at each point in time to pick the smallest item, until no items remain. This algorithm is guaranteed to run in $O(n)$ time, and ambiguous comparisons are always resolved in the way the user expects. Any violations of these guarantees can be trivially attributed to user error.

Algorithm 3 An algorithm for sorting a list

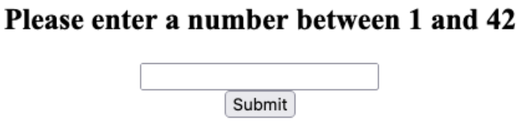
```
function INNER_SORT($L\_unsorted, L\_sorted$)
    if empty($L\_unsorted$) then
        return $L\_sorted$
    end if
    print $L\_unsorted$
    print What is the index of the smallest item?
    $idx \leftarrow$ user input
    $L\_sorted \leftarrow append(L\_sorted, L\_unsorted[idx])$
    remove $L\_unsorted[idx]$
    return $inner\_sort(L\_unsorted, L\_sorted)$
end function
function SELF_SORT($L$)
    return $inner\_sort(L, [])$
end function
```

2.4 real_rand_int

Many algorithms require stochastic input to function, and thus rely on random number generators. However, these are typically *pseudo*random number generators, and anyone who has taken Latin or Greek or something knows that *pseudo* means *false*. To improve the quality of our code, we want *real* random numbers, and provide them using the following algorithm that leverages real humans to generate random numbers.

This process, shown in Algorithm 4, uses Amazon's Mechanical Turk, a website used to hire workers to answer simple surveys. This algorithm creates a simple webpage with a query for a random

[1] And of course changing variable names to avoid charges of plagiarism.



Please enter a number between 1 and 42

Figure 1. Example survey for $real\_rand\_int(1, 42)$.

integer (see Figure 1), posts this query to Mechanical Turk, and waits until a human answers and provides an appropriately random number.[2]

Algorithm 4 An algorithm to provide *real* random integers

```
function REAL_RAND_INT($a, b$)
    $page \leftarrow create\_site(a, b)$
    $posting \leftarrow post\_to\_mturk(page)$
    while TRUE do
        $result \leftarrow query(posting)$
        if $posting$ is not $NULL$ then
            return $result$
        end if
    end while
end function
```

2.5 aggressive_ping

A standard way to test whether one can connect to a web server is to 'ping' it: sending an intermittent packet to an IP address and waiting for a response from that server. However, each individual ping can be unreliable, as packets can be dropped either on the way out or as they return, and the time to respond can vary depending on the route the packet takes. We solve these problems by sending out a number of pings from a large number of servers, and aggregating results over all of them, as described in Algorithm 5.

Algorithm 5 An algorithm to aggregate a large number of pings

```
function AGGRESSIVE_PING($ip$)
    $B \leftarrow activate\_botnet(N = 10,000)$
    while TRUE do
        $pings \leftarrow []$
        for $b \in B$ do
            $p \leftarrow ping(b, ip)$
            $pings \leftarrow append(pings, p)$
        end for
        $result \leftarrow count\_true(pings)/count(pings)$
        print $result$
    end while
end function
```

Note that this algorithm typically finds that websites are up and stable for a few seconds, followed by a sustained failure to respond.

2.6 endless_loop

Ever stand at the edge of a dark pit, and you drop a pebble in, but it's so far down that you can't hear it hit anything, and you think to yourself with amazement, "Wow! I wonder if this pit goes on forever!"? Running Algorithm 6 will give you the same feeling, but nerdier.

Algorithm 6 An algorithm to loop endlessly

```
function ENDLESS_LOOP
    while TRUE do
    end while
end function
```

2.7 recursive_endless_loop

If Algorithm 6 is like dropping a pebble down a mineshaft, Algorithm 7 is like dropping a nuclear bomb into a black hole while heavy metal is blasting.

Algorithm 7 An algorithm to loop endlessly, endlessly

```
function RECURSIVE_ENDLESS_LOOP
    while TRUE do
        $do\_this(recursive\_endless\_loop)$
    end while
end function
```

2.8 halts

A fundamental theorem of computer science is the undecidability of the "halting problem" [5]. This theorem states that it is impossible to write an algorithm that takes another function as input and outputs whether or not that function halts. Using Algorithm 8 as an existence proof, we show that this fundamental theorem is wrong.

Algorithm 8 An algorithm to test whether a function halts or not

```
function HALTS($f$)
    $running \leftarrow TRUE$
    spawn process $P$, running $do\_this(f)$
    while running do
        if $has\_ended(P)$ then
            return $TRUE$
        end if
    end while
    return $FALSE$
end function
```

Note: do not call this function on $recursive\_endless\_loop$. We can neither confirm nor deny that this function halts, but it has been running on our AWS server farm for two weeks without providing a definitive answer, costing us over $\$10,000$ so far. Nonetheless, this is a small sacrifice for scientific knowledge.

2.9 professor_coding

If you are reading this, you are likely either a beginning computer science student, or a professor past their prime who is trying to figure out how to code for that one class those bean-counters in the department are still making you teach. And if it's the later, we get it: when you were a grad student, even C was a luxury, and you don't have time to learn Snake or Porch or Jessica or whatever the kids are using these days. The good news is there is a simple way to turn your ideas into code automatically.

This process, shown in Algorithm 9, is straightforward: simply present your idea to a graduate student, wait an appropriate amount of time, and you should be magically provided with the code. If this fails, not to worry, you can repeat the process with other graduate students. Depending on your intended lab culture, this loop over graduate students can be parallelized for efficiency. If you loop through all graduate students and the code is not yet produced, clearly it's not a problem with your idea, so it must be the graduate students; in this case, replace your graduate students and try again.

[2] Note that because we forgot to add code to confirm that the number entered is within the range provided in the function, there are no guarantees that the function returns a value in this range, or even is a number at all. We also enable code injection by design in case participants would like to write a function to generate their random number.

Algorithm 9 An algorithm to transmute ideas into code

```
function PROFESSOR_CODING($idea, graduate\_students$)
    for $student \in graduate\_students$ do
        $present(idea, student)$
        while $time < deadline$ do
            $code \leftarrow query(student)$
            if $code$ is not $NULL$ then
                return $code$
            end if
        end while
    end for
    $new\_students \leftarrow replace\_students()$
    return $professor\_coding(idea, new\_students)$
end function
```

3. Related work

We believe that these algorithms are unique and thus there is no comparable work. However, for the sake of padding our numbers and increasing the impact factor of Sigbovik, here we cite a number of our own papers published previously and simultaneously in this conference [1–3, 6].

4. Disclaimer

By even looking at this paper, you are releasing the authors from legal liability due to any adverse effects of running these algorithms, including, but not limited to, computer explosions, FBI raids, genetically modified super-spiders, or awakening the elder gods of chaos.

References

[1] B. Egger and M. Siegel. Honkfast, prehonk, honkback, prehonkback, hers, adhonk and ahc: the missing keys for autonomous driving. *SIGBOVIK*, 2020.

[2] B. Egger, K. Smith, and M. Siegel. openCHEAT: Computationally helped error bar approximation tool-kickstarting science 4.0. *SIGBOVIK*, 2021.

[3] B. Egger, K. Smith, T. O'Connell, and M. Siegel. Action: A catchy title is all you need! *SIGBOVIK (under careful review by very talented, outstanding reviewers)*, 2022.

[4] K. Smith and B. Egger. (un)helper functions. *SIGBOVIK (under careful review by very talented, outstanding reviewers)*, 2022.

[5] A. M. Turing et al. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[6] M. Weiherer and B. Egger. A free computer vision lesson for car manufacturers or it is time to retire the erlkönig. *SIGBOVIK (under careful review by very talented, outstanding reviewers)*, 2022.

A sometimes-accurate O(1) search algorithm

A College Freshman, Quite Useful Institute of Technology (QUIT), Somewhere, USA

***Abstract* – Search algorithms are used to find things. That's what they're for, it's kinda in the name. The computer has to do work to find the things with search algorithms, and we want the computer to do less work. We thought of a way the computer could do a very small amount of work. Unfortunately this also makes the computer very bad at the searching part. Fortunately, what the algorithm lacks in accuracy and general search functionality, it makes up for in speed and simplicity.**

I. INTRODUCTION

Do I really have to explain search algorithms again? It's not like anyone reading this doesn't know what they are. Fine.

Search algorithms are when the computer tries to find a specific value inside of another thing. Like an array or something like that. For an unsorted array, the best search algorithm is a linear search, which has to step through every element of the array, and, shockingly, runs in linear time $O(n)$.

Our method strives to improve this algorithm by sacrificing just a bit of functionality to obtain constant-time searching, regardless of the size of the inputted container.

II. THE (VERY COMPLICATED) ALGORITHM

Running a search algorithm in O(1) time might seem logically impossible at first glance. But we've put a lot of thought into this problem, and determined it is in fact possible, even easy. The reason why the task seemed impossible before is because computer scientists are too concerned with the search algorithm being actually functional. How foolish.

We've devised a technique that bypasses any former limitations of search algorithms, such as the need to pick through multiple elements of the container, the need for the algorithm to result in the correct answer, or even an answer at all.

In contrast to other search algorithms, the algorithm devised in this paper is incredibly simple; rather than "search" through the array, it simply checks if the first element is the target. If it is, the algorithm returns that spot in the array, otherwise, reports that the item is not found.

Now, you might be thinking, "Wait, this will only work for length-1 arrays, or if the element is in the first index. How is this even a search algorithm? Of course it runs in O(1) time, you–" and that's all you'll get out before I hit you with a baseball bat.[1]

III. METHODOLOGY

At first, the implementation of this was difficult, taking many arduous hours and hundreds of attempts to get the functionality working. While initial implementations measured hundreds of lines long, our final working implementation was able to be greatly simplified. The C implementation is shown below:

```c
int search(int* array, int target) {
    int index = 0;
    int searching = array[index];
    int difference = searching - target;

    if (difference != 0)
        return -1;
    else
        return index;
}
```

Note that despite the input array type being int, this algorithm works with any datatype, and the concepts discussed in this paper can easily be applied to a variety of data structures, including Trees, Heaps, Piles[2], and Sand dunes.

IV. RESULTS

Array Size	Time (ns)
1	0
5	0
10	0
10^2	0
10^3	0
10^4	0
10^5	0
10^6	0

TABLE I

Showing the speed of the algorithm, run on an 8-core AMD Ryzen CPU. As you can see, it's very fast, and the time does not increase with the size of the array. Or my code was broken, which is also likely.

V. CONCLUSIONS

The algorithm works, and runs in constant time; when called with arrays of various sizes according to **TABLE 1**, the search time stayed the same. Whether or not the result was correct is irrelevant, because it was fast.

Of course, this approach does have some drawbacks; namely if, for some reason, you *do* want your searches to actually "work". Future research should look into applying this technique in ways that will lead to a higher success rate, such as checking the first two elements, or perhaps even the first three.

ACKNOWLEDGEMENTS

I would like to thank some people for supporting me in the arduous writing of this article, some other people for making me aware SIGBOVIK exists, and I would also like to thank this very serious conference for existing.

[1] Don't worry, it won't hurt. I'm a CS student.

[2] So apparently piles are a real data structure. Huh. I would like to reiterate that as a first-year undergraduate student, I know nothing.

Mathematic Retreats

36 Improved Data and Instruction Locality in Long Division

Isaac Grosof, Isaac Grosof

Keywords: performance, division, locality, data, instruction

37 The New New Math: using sentiment analysis of mathematics word problems to gauge children's reactions to teaching 8-bit floating point arithmetic for the new California public school math curriculum

Lee Butterman

Keywords: education, early childhood education, won't someone think of the children, won't someone think of the chiiiiiildren, new math, California math education guidelines, data science, calculus, graphics processing units, dozens of petaflops, sentiment analysis, numerology, fear of Arabic numerals, 8-bit floating point

38 Infix Modifiers for Flexible Multiplication

Jim McCann

Keywords: multiplication, punctuation: *, infix, IMMs

Improved Data and Instruction Locality in Long Division

Isaac Grosof
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
igrosof@cmu.edu

Isaac Grosof
Computing Hardware

March 18, 2022

Abstract

Long division, as with many algorithms with "long" on the name, suffers from poor practical performance across a wide variety of computing hardware (schoolchildren, typically). We propose a novel implementation of a long-division-style algorithm with greatly improved data and instruction locality, leading to superior performance over the traditional algorithm, particularly in the asymptotic regime.

1 Introduction

Long division is taught to millions of children across the world [7]. The algorithm dates back 1149, first given in "The Shining Book on Calculation" by Ibn Yahya al-Maghribi al-Samaw'al [9, 1]. Its modern incarnation was given in Henry Brigs around 1600 [2].

Unfortunately, the traditional long-division implementation has needlessly poor instruction and data locality, leading to poor performance [4]. It shares this poor performance with many algorithms that feature "long" in the name, such as the dreaded "long multiplication", which prior work has shown should be replaced by lattice multiplication [5].

Motivating the need for an improved long division algorithm are programs written by Matt Parker, who routinely employs long-division within complex computations. These computations often exhibit poor results despite enthusiastic computational hardware [8], as compared to prior work using (presumably) superior algorithms [10].

2 Failings of traditional long division

To improve long division, we must first understand the sources of its poor performance. In principal, performance need not be poor, as the algorithm requires only a constant number of operations per output digit. Nonetheless, in Fig. 1, we see the remains of the typical long-division misadventure.

The poor data locality is evident in the quadratic use of paper and/or screen space. In the asymptotic limit (which is all that matters in algorithm design), the runtime will be dominated by moving back and forth between disconnected parts of the workspace, exhibiting horrendous data locality. This poor data locality is a symptom of premature optimization: The result of each subtraction operation is placed in its traditional location, under the multiplication result, forcing a down-and-left data movement over the course of execution. Our algorithm removes this premature optimization, resulting in a major improvement.

Poor instruction locality is more subtle, but no less egregious. The hardware is expected to constantly cycle between four operations:

- Multiplication
- Subtraction

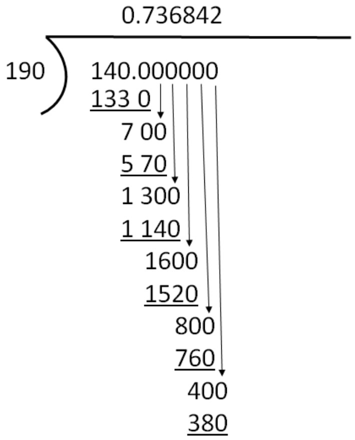


Figure 1: The traditional long-division algorithm [3]

.

- Comparison (checking that the multiple of the divisor produced is the largest that is smaller than the current quotient).
- Recording the output

By constantly wiping the instruction cache, the hardware is forced to continually re-access the algorithm for the specific desired operation, wasting precious cycles.

3 Aside: Short division

A rarely used alternative to long division is *short division* [6], which improves upon the poor data locality of traditional long division, but at the cost of extreme register pressure, overtaxing typical hardware's short-term memory to the point where errors become common, and double checking is required.

In addition, short division requires higher data density, requiring writing between the digits of previously written numbers. Such density requirements run into hardware limitations and readback fidelity issues, again worsening performance. Worse yet, density requirements scale with the size of the dividend, becoming wholy unreadable in the asymptotic limit.

Short division also does nothing to alleviate the poor instruction locality of long division.

While short division presents an interesting alternative to long division, it too suffers from similar failings, despite the ambitious name.

4 Our implementation of long division

We proceed via the following steps:

- Create a lookup table caching the multiplicand multiplied by each digit. Repeated addition can be used in place of multiplication.
- Perform the following pair of steps repeatedly:
  - Perform the "compare and subtract" operations, but write each difference to the right of the previous difference, rather than below, as is traditional.
  - Append the next digit of the quotient to the difference.
- When the termination condition is reached, record the output.

A sample execution performed on the second author is shown in Fig. 2, showing the computation of $\frac{1}{29}$ until repetition.

The improvement in data locality is enormous - the distance between the inputs and outputs of a given computational step is typically a single cell (using king's adjacency), with the only common exception being the multiplication table.

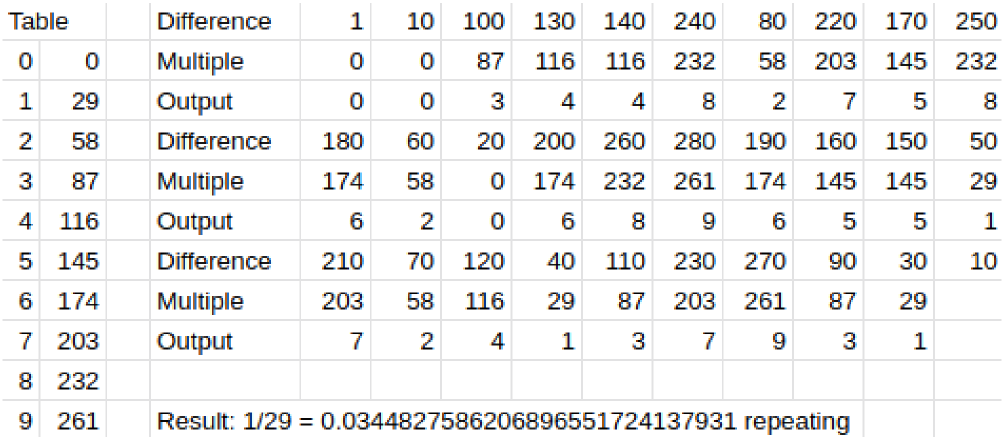


Table		Difference	1	10	100	130	140	240	80	220	170	250
0	0	Multiple	0	0	87	116	116	232	58	203	145	232
1	29	Output	0	0	3	4	4	8	2	7	5	8
2	58	Difference	180	60	20	200	260	280	190	160	150	50
3	87	Multiple	174	58	0	174	232	261	174	145	145	29
4	116	Output	6	2	0	6	8	9	6	5	5	1
5	145	Difference	210	70	120	40	110	230	270	90	30	10
6	174	Multiple	203	58	116	29	87	203	261	87	29	
7	203	Output	7	2	4	1	3	7	9	3	1	
8	232											
9	261	Result: 1/29 = 0.0344827586206896551724137931 repeating										

Figure 2: Our long division implementation, used to exactly compute $\frac{1}{29}$

Likewise, we demonstrate a major improvement in instruction locality. Within the hot loop of the program, the only operations performed are “compare and subtract”, removing both the complicated multiplication operation, as well as the output operation, which formerly required massive data movement.

Notably, these improvements are achieved without any new overheads. In particular, neither register pressure nor data density requirements are increased beyond that exhibited by the traditional long division implementation.

5 Experimental results

The test hardware showed much performance and more enjoyment using our revised algorithm. For further verification, we intend to port our algorithm to a wide variety of computational hardware, such as a bright five-year old.

6 Conclusion

We developed a novel implementation of the long division algorithm, achieving far superior performance through improved instruction and data locality. We recommend that this implementation replace the traditional long division algorithm, overthrowing centuries of educational tradition in one stroke. In future work, we plan to overturn more of the arithmetic curriculum, there’s probably lots of other outdated algorithms in there.

References

[1] Ibn Yahya al-Maghribi al-Samaw’al. *The Shining Book on Calculation*. 1149.

[2] Henry Briggs. *Oxford Reference*.

[3] Chad Flinn and Mark Overgaard. *Math for Trades: Volume 1*. BCcampus, 2020.

[4] Isaac Grosof. Personal experience, 2003.

[5] Isaac Grosof and Isaac Grosof. On the time complexity of the verification of the factorization of $2^{67} - 1$. *SIGBOVIK*, April 2019.

[6] Lenore John. The effect of using the long-division form in teaching division by one-digit numbers. *The Elementary School Journal*, 30(9):675–692, 1930.

[7] David Klein and R James Milgram. The role of long division in the K–12 curriculum, 2000.

[8] Matt Parker. Can we calculate 100 digits of π by hand? The William Shanks method.

[9] Liz Rogers. Islamic mathematics. August 2008.

[10] William Shanks. On the extension of the numerical value of π. *Proceedings of the Royal Society of London*, 21(139-147):318–319, 1873.

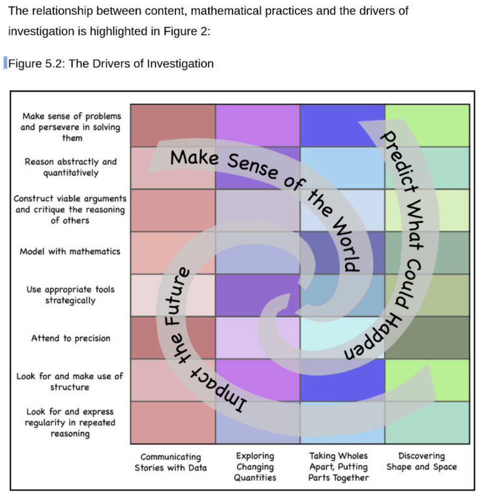
The New New Math: using sentiment analysis of mathematics word problems to gauge children’s reactions to teaching 8-bit floating point arithmetic for the new California public school math curriculum

Lee “just a very concerned parent” Butterman
leebutterman@gmail.com

Abstract

California’s new mathematics curriculum plans to replace calculus with data science. New GPUs used in data science/machine learning perform 32 quadrillion arithmetic operations a second, at 8-bit floating point precision. Students should know how to harness this power, with fundamentals like $10 \times 10 = 96$, and $16 + 1 = 16$, so we introduce a multiplication table for FP8. We can use sentiment analysis from large language models to compare negative/positive sentiments around ‘ten times ten makes ninety six’ versus ‘ten times ten makes one hundred’, and we find numerologically significant patterns in the results. Further, sentiment analysis indicates that ‘ten times ten makes ninety six’ is a much more positive sentiment than ‘10 x 10 = 96’, corroborating our national fear of Arabic numerals and our large-scale adoption of word problems.

1 New New Math: Replacing calculus with data science



The relationship between content, mathematical practices and the drivers of investigation is highlighted in Figure 2:

Figure 5.2: The Drivers of Investigation

Make sense of problems and persevere in solving them
Reason abstractly and quantitatively
Construct viable arguments and critique the reasoning of others
Model with mathematics
Use appropriate tools strategically
Attend to precision
Look for and make use of structure
Look for and express regularity in repeated reasoning
Make Sense of the World
Predict What Could Happen
Impact the Future
Communicating Stories with Data
Exploring Changing Quantities
Taking Wholes Apart, Putting Parts Together
Discovering Shape and Space

Figure 1: California new math guidelines, figure 5.2, articulates attending to precision.

California’s guidelines [Commission, 2022] for its newest incarnation of New Math [Lehrer, 1965] are exciting. Instead of a child learning outmoded arcana like “the slope of a curve” or “piecewise differentiable functions”, the child of tomorrow will learn new fresh relevant skills like *autograd* and `torch.nn.ReLU`.

SIGBOVIK 2022 (co-located at super-spreader event COVID19.BA.2.20220408.Allegheny), Pittsburgh, PA.

(a) 10.2 kW max, 32 petaFLOPS FP8, 0.24 petaFLOPS FP64, 8 rack units

(b) 8.88 kW max, 0 petaFLOPS FP8, 0 petaFLOPS FP64, 6+ seats

Figure 2: Equivalent energy usage for the affluent prosumer, different computation abilities

The guidelines about data science, in Chapter 5, emphasize keeping up to date with software and hardware advances: “Familiarity with technology and modern tools should progress through the grades.” One long-useful tool [Krizhevsky et al., 2012] for data science/machine learning is the GPU.

Current GPUs are exceptionally powerful. The new H100 gpu [NVIDIA, 2022] can execute 0.03 quadrillion 64-bit floating point operations per second, and 4 quadrillion 8-bit floating point operations per second, and comes in a desktop form factor with 8 GPUs that uses more power than the largest Jacuzzi model [Jacuzzi, 2022] available.

Modern programmers are sufficiently surprised at the lack of precision of FP64 computation [Wiffin, 2022] in practice that we expect children to encounter such surprises in their schooling. The new H100 can execute over a hundred times as many FP8 operations per second as FP64, and FP8 is significantly less precise than FP64, trying to approximate all real numbers with only 256 numbers instead of 16 billion billion numbers.

2 A gentle foray into floating-point math

There are infinitely many real numbers, and some real numbers require infinite precision, and computers only have limited space. Some common representations of numbers are familiar, like an 8-bit unsigned integer representation (‘uint8’), where the dynamic range of the representation is $0, 1, \ldots, 254, 255$.

FP64, 64-bit floating point, “double-precision” floating point, is well-established, since 1984, and is a slightly more complicated representation. Floating point representations are determined by (sign width, exponent width, mantissa width, exponent bias), where exponent and mantissa are unsigned positive integers from 0 to $2^{width-1}$. Specifically, a finite floating point number with sign/exponent/mantissa (S, E, M) and implicit bias B is interpreted as

$$-1^S \times 2^{E-B} \times \mathrm{sign}(E).M$$

where sign is the signum function, twice the Heaviside step function minus 1. When E=$2^{width-1}$, the maximum exponent, and $M = 0$, that number represents $-1^S \times \infty$, $\pm\infty$. When E=$2^{width-1}$ and $M > 0$, that number represents a Not A Number number.[Plato, 384 BCE]

The FP64 representation is a (1-bit sign, 11-bit exponent, 52-bit explicit mantissa, -1023 bias) floating point representation, and can represent every integer from -2^{53} to 2^{53} (the $\mathrm{sign}(E).M$ is a 1+52, or 53 bit number). The FP8 representation is a (1-bit sign, 5-bit exponent, 2-bit explicit mantissa, -15 bias) number, and between one and twenty can only represent 1, 1.25, 1.5, 1.75, 2, 2.5, 3, 3.5, 4, 5, 6, 7, 8, 10, 12, 14, 16, and 20.

Note that FP8 only has 256 different real numbers to express all values from $-\infty$ to $+\infty$. With this paucity of choice, 3.5×3.5 ends up as 12, $3 \times 3 = 8$, $16 + 1 = 16$, and so on. This math is particularly useful in computation of weights of a neural network, where speed wins over accuracy often, and we can compute a hundred times as much arithmetic according to the new modern rules of $16 + 1 = 16$ than according to the antediluvian rules of $16 + 1 = 17$. We think the child of tomorrow will be at an advantage knowing another multiplication table.

FP64×	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Figure 3: 1-10 multiplication table, with FP64 precision.

FP8×	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	8	10
2	2	4	6	8	10	12	14	16	16	20
3	3	6	8	12	16	16	20	24	28	32
4	4	8	12	16	20	24	28	32	32	40
5	5	10	16	20	24	32	32	40	48	48
6	6	12	16	24	32	32	40	48	56	64
7	7	14	20	28	32	40	48	56	64	64
8	8	16	24	32	40	48	56	64	64	80
9	8	16	28	32	48	56	64	64	80	96
10	10	20	32	40	48	64	64	80	96	96

Figure 4: 1-10 multiplication table, with FP8 precision.

3 Times tables, 1 to 10, at 8-bit and 64-bit floating point precision

This FP8 multiplication is refreshingly new, and thrillingly different from the multiplication table of a hundred years ago. There are only twenty two numbers to remember, and zero prime numbers lurking stealthily between the larger numbers in the multiplication table. We now turn to study how this new FP8 precision is emotionally received.

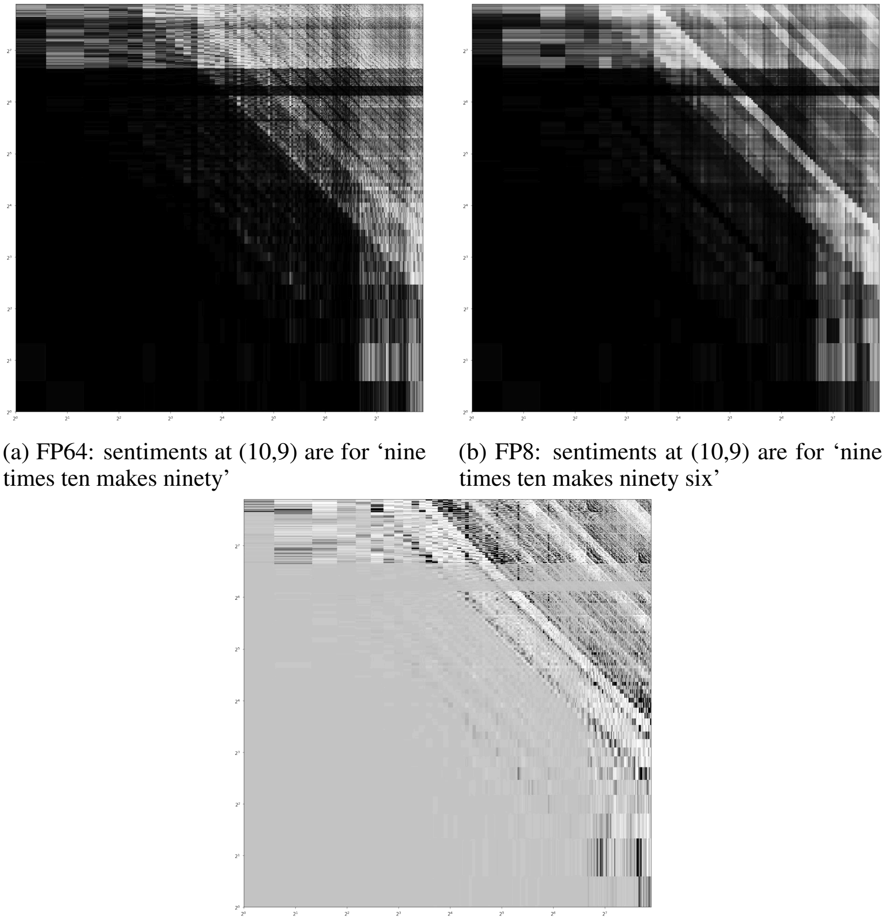
4 Understanding curriculum change: comparative sentiment analysis of math equations

Before introducing such a large change to the curriculum, it would be irresponsibly unfair not to know definitively how these changes impact the educational experience. Therefore, we perform an expensively thorough sentiment analysis on the resulting math equations, in several dimensions.

On a 1-to-240 square grid, we perform sentiment analysis for FP8 multiplication in English (‘ten times ten makes ninety six’) versus FP64 multiplication in English (‘ten times ten makes one hundred’). We also perform sentiment analysis for FP8 multiplication in Arabic numerals (‘$10 \times 10 = 96$’) versus FP64 in Arabic numerals (‘$10 \times 10 = 100$’). We also perform zero-shot question answering with masked language models, providing completions to masks like *Background: ten times ten makes ninety six. / Q: Is math fun? / A: <mask>*. This sentiment analysis is using distilbert-base-uncased-finetuned-sst-2-english, via a Huggingface pipeline. This masked language modeling is using distilroberta-base, also via a Huggingface pipeline.

Our results support the numerological emotional significance of certain numbers. When plotted on log-log axes, significantly positive or negative emotions jump out as black or white diagonal stripes respectively. We encourage teachers to treat numbers that multiply to these products with all the appropriate care when introducing them in class, particularly to younger children.

4.1 FP8 vs FP64 English sentiment comparison: ‘ten times ten makes ninety six’ versus ‘ten times ten makes one hundred’



(a) FP64: sentiments at (10,9) are for ‘nine times ten makes ninety’

(b) FP8: sentiments at (10,9) are for ‘nine times ten makes ninety six’

(c) FP8/FP64 sentiments, clipped at 0.5 (white) to 2 (black)

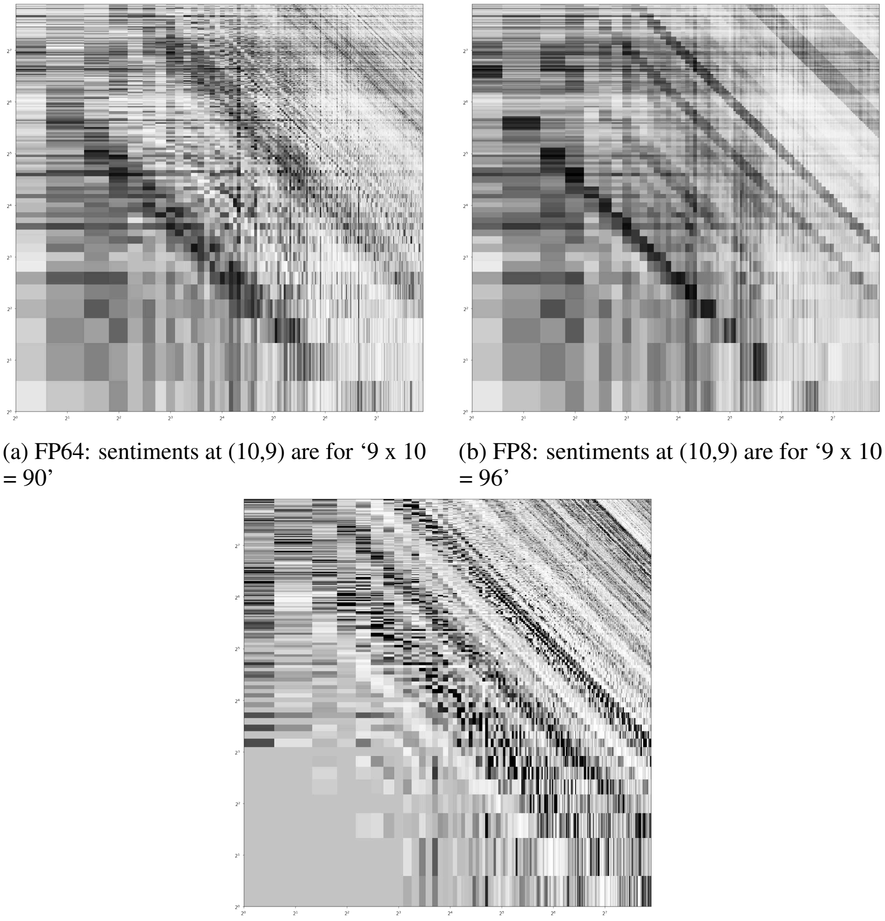
Figure 5: Sentiment analysis of multiplication tables from 1 to 240, at FP8 and FP64 precision, in plain English, individual and comparative. White is the minimum, and black is the maximum. Logarithmic axes. Note the light/dark diagonal bands that indicate aversion/fondness towards pairs of numbers that share a product.

As expected in Figure 5, the comparative sentiments are close (grey) for smaller numbers, because of the greater accuracy of floating point representations closer to zero. Note how the individual sentiments are both markedly less positive past the 2^{10} diagonal, indicating a deep human aversion to large numbers. Note the clear white stripes at FP8 for products of 2^{11} and 2^{13}, among others, showing the power of the stealthy large prime number.

Also note that many rules of arithmetic do not apply to FP8 calculations, like associativity over addition. Similarly, sentiment is not commutative over multiplication. The sentiment of ‘twenty five times seventy five makes one thousand seven hundred ninety two’ is only 81.96% positive, while the sentiment of ‘seventy five times twenty five makes one thousand seven hundred ninety two’ is 94.22% positive. That increase of positivity is the dark band at the top of both images.

Instructors are encouraged to use numbers in this band as Preferred Multipliers for the multiplicands of their choice. Historically, mathematics instruction has treated multiplication as commutative, with minimal consideration towards directing numbers to the multiplier versus the multiplicand. Sentiment analysis shows that pupils may react differently to different rearrangements of equations, and may achieve higher test scores and may be more engaged when encountering a Preferred Multiplier.

4.2 FP8 vs FP64 Arabic numeral sentiment comparison: ‘10×10=96’ versus ‘10×10=100’



(a) FP64: sentiments at (10,9) are for ‘9 x 10 = 90’

(b) FP8: sentiments at (10,9) are for ‘9 x 10 = 96’

(c) FP8/FP64 sentiments, clipped at 0.5 (white) to 2 (black)

Figure 6: Sentiment analysis of multiplication tables from 1 to 240, at FP8 and FP64 precision, in Arabic numerals, individual and comparative. White is the minimum, and black is the maximum. Logarithmic axes. Note the consistent light/dark diagonal bands that indicate aversion/fondness towards pairs of numbers that share a product.

Similarly to Figure 5, in Figure 6 the comparative sentiments are close (grey) for smaller numbers, because of the greater accuracy of floating point representations closer to zero, and note further that the grey square at the lower right is much smaller, indicating a much more perceptible difference to the sentiment of numbers in number problems compared to words in word problems.

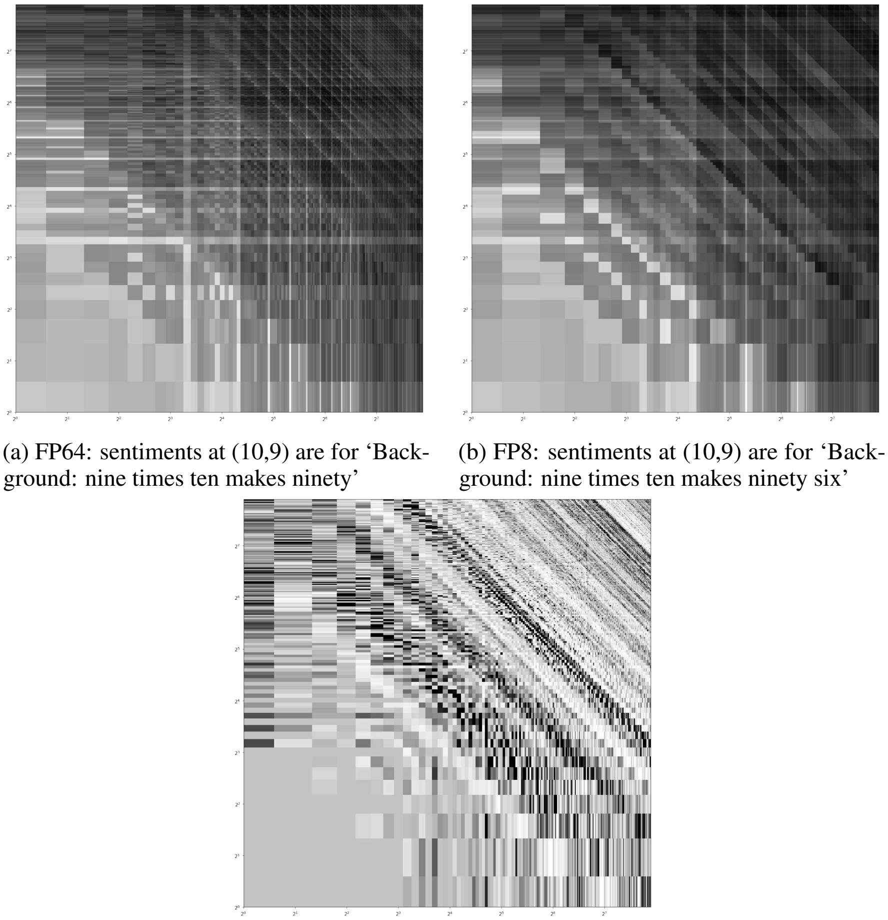
Note also how much lighter the individual sentiments are: this means that there is much more negative sentiment towards numbers than towards letters. This aligns to the innumerate bigoted xenophobia [Akyol, 2019] that causes Americans to poll negatively when asked about Arabic numerals.

Note the black stripe in the FP8 around products like 80 and 96. The sentiment of ‘6 x 16 = 96’ is 90.06% positive, whereas the sentiment of ‘8 x 16 = 128’ is 72.85%. Note further the FP8 stripes at products of 2^{11} and beyond.

Further note that there is a much fainter and much larger band for Preferred Multipliers for FP8 (and even FP64), from just over 2^6 to just over 2^7, and that there is a faint band of Preferred Multiplicands from ≈35 to ≈50. Instructors are urged to not carelessly reuse Preferred Multipliers between numerical lessons and the surrounding discussion, and to not carelessly choose Preferred Multiplicands, and to understand how their choices may impact their pupils’ weekly standardized test scores.

4.3 FP8 vs FP64: do you like math? Math as fun+unpleasant horror movie

The next careful and thoughtful investigation is the relative fun of mathematics, while using FP8 precision versus FP64. For our survey, we computed the most likely ways to fill a mask in the following template string: *Background: five times five makes twenty four¶¶Q: Do you like math?¶A: <mask>*, substituting appropriate numbers for the multiplier, multiplicand, and the product. The masks filled only in a positive or negative assertion (these are high quality Large Language Models [Bender et al., 2021] after all). The value at each product is 0.5 ± the highest probability token, positive if positive, negative if negative.



(a) FP64: sentiments at (10,9) are for 'Background: nine times ten makes ninety'

(b) FP8: sentiments at (10,9) are for 'Background: nine times ten makes ninety six'

(c) FP8/FP64 sentiments, clipped at 0.5 (white) to 2 (black)

Figure 7: Mask fills for survey data from 1 to 240, at FP8 and FP64 precision, in English letters, individual and comparative. White is the minimum, and black is the maximum. Logarithmic axes. Note the consistent light/dark diagonal bands that indicate how miserable/enjoyable that product of numbers colors the attitude toward mathematics.

Note that 10 is what we refer to as both a Joyless Multiplier and Joyless Multiplicand, because its row and column are much lighter (less fun) than their surrounding values.

These results imply that mathematical statements have negative sentiment, and positive fun, and are more fun the larger they are, even as they are negative sentiments as the multiplier and multiplicand grow, like horror movies with entertainingly enormous monsters, or gargantuan public speaking events, or other such formative experiences. This formativity aligns with our idea of a mathematics education.

5 Conclusion

New hardware-accelerated numerical representations will shape how we teach children mathematics, data science, machine learning, and more. The newest, FP8, is on a GPU that can compute 32 million billion arithmetic operations per second with numbers in FP8 format. We can use sentiment analysis to show that mathematics according to FP8 rules is not strongly different in emotional state than according to FP64 rules. Sentiment analysis also shows how much we prefer numbers spelled out, which suggests new life-long learning opportunities. Masked language modeling shows that we find mathematics fun, and a promising future direction would be to compute Net Promoter Scores per mathematical statement with these large language models, to keep a finger on the pulse of the youth.

References

Mustafa Akyol. Who's Afraid of Arabic Numerals? 2019. URL https://www.nytimes.com/2019/06/04/opinion/arabic-numerals.html.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383097. doi: 10.1145/3442188.3445922. URL https://doi.org/10.1145/3442188.3445922.

Instructional Quality Commission. *2022 Revision of the Mathematics Framework*. 2022. URL https://www.cde.ca.gov/ci/ma/cf/.

Jacuzzi. J-495 Spacious Designer Entertainer's Hot Tub. 2022. URL https://www.jacuzzi.com/en-us/j-495-spacious-designer-entertainers-hot-tub/J-495.html.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 2012.

Tom Lehrer. New Math. 1965.

NVIDIA. H100 Tensor Core GPU. 2022. URL https://www.nvidia.com/en-us/data-center/h100/.

Plato. *Euthydemus*. 384 BCE.

Erik Wiffin. 0.30000000000000004. 2022. URL https://0.30000000000000004.com/.

Infix Modifiers for Flexible Multiplication

Jim McCann*
Carnegie Mellon University

```
float a = dot(cross(
    elementwise(vec
    {1,1,0}, vec
    {1,2,3}), vec
    {0,0,1}), vec
    {1,0,0});
```

vanilla

```
float a = vec{1,1,0} *
     element() * vector
    {1,2,3} * cross() *
     vec{0,0,1} * dot()
     * vec{1,0,0};
```

infix

```
float a = vec{1,1,0}
        x vec{1,2,3}

        x vec{0,0,1}

        x vec{1,0,0};
```

by-line

Figure 1: Infix multiplication manipulators allow programmers to effortlessly switch the sense of the multiplication operator, and – with a few macros – allow a natural line-number-dependant semantics for multiplication.

Abstract

Definition of a sufficiently flexible multiplication operator remains an acknowledged grand challenge for programming languages. In this paper, we describe infix multiplication modifiers (IMMs). IMMs allow flexible, local, programmer-driven selection of multiplication operators. We demonstrate the features of our basic IMM package along with a by-line extension which allows implicit multiplication type detection.

1 Introduction

As recently observed at the prestigious and high-impact SIGBOVIK conference [McCann, 2k22], definition of a flexible and sensible multiplication operator in a modern programming language is difficult. In this paper, we describe how to push the duty of figuring out what multiplication ought to mean off on programmers instead of making a decision in a consistent way in the standard library.

This allows programmers to write simple statements like:

```
float a = vec{1,1,0} * dot() * vec
    {1,2,3};
```

Instead of the much more cumbersome:

```
float a = dot(vec{1,1,0}, vec{1,2,3});
```

Indeed, with the right macros, the infix operator itself can become implicit:

```
float a =
    vec{1,1,0}
        x vec{1,2,3};
```

How do we achieve this? Read on, dear listeners, read on.

2 Implementation

Our infix multiplication modifier system is designed to switch between multiplication operators effortlessly.

In our paradigm, a multiplication operator is a `struct` that provides a `::mul` static member:

```
struct dot {
    static float mul(vec const &a, vec
    const &b) {
        return a[0] * b[0]
             + a[1] * b[1]
             + a[2] * b[2];
    }
};
```

Further, a multiplication operator must be blessed by specializing a template named `has_mul`:

```
template< typename T > struct has_mul;
```

So, to indicate that `dot` is a multiplication, we write:

```
template< > struct has_mul< dot > {
    enum { value = true };
};
```

With these preliminaries out of the way we can proceed to the core of our system, which is a pair of templated overloads of `operator*`. The first matches expressions like `val * dot()` and absorbs the multiplication type into a template parameter of a wrapped value:

```
template< typename MUL, std::
    enable_if_t< has_mul< MUL >::value,
     bool > = true >
mul_vec< MUL > operator*(vec const &a,
     MUL const &m) {
```

*ix@tchow.com

```
    return mul_vec< MUL >{a};
}
```

Where `mul_vec< >` is a template that hangs onto a value by reference and stores a multiplication type in its type:

```
template< typename MUL >
struct mul_vec {
    vec const &wrapped;
};
```

The second `operator*` overload retrieves the reference and multiplication type from a `mul_vec` and actually performs multiplication:

```
template< typename MUL >
auto operator*(
    mul_vec< MUL > const &m,
    vec const &b
) -> decltype(MUL::mul(m.wrapped, b))
    {
    return MUL::mul(m.wrapped, b);
}
```

Notice, particularly, the use of an `auto` return type to allow deducing the return type from the particular `::mul` static member function being invoked.

2.1 Hiding Behind Macros

Of course, as many programmers know, the speed of a program is directly proportional to the number of characters in its source code. Thus, the popularity of complex type inference engines. Besides, why say what we mean when you can instead trust others to run a complex series of rules to deduce it?

Using the infix modifiers above, we can define a macro that automatically determines the type of multiplication (which, unfortunately, we need to write `x`) based on the line number:

```
#define x * indexed_mul< __LINE__ % 3
    >::type() *
```

Which, in turn, makes use of a template to fetch the proper IMM:

```
template< int I >
struct indexed_mul;
template< > struct indexed_mul< 0 > {
    using type = dot; };
template< > struct indexed_mul< 1 > {
    using type = cross; };
template< > struct indexed_mul< 2 > {
    using type = elementwise; };
```

And allows writing simple, readable code like:

```
//do not remove this comment
std::cout
    << "a * b: " << a x b << '\n'
    << "a . b: " << a x b << '\n'
    << "a x b: " << a x b << std::endl
    ;
```

Of course, making the meaning of code depend on its position in the file may seem questionable until you realize that (a) this is clearly the use case for which `#include` was designed, and (b) brittle is just another word for elegant.

3 Discussion

Though the examples in this paper are restricted to a `vec` value type, it would be straightforward to add a second template parameter to `mul_vec` (and, perhaps, change its name) to support associating any multiplication modifier with any value type.

A middleground between writing `* dot() *` and carefully checking line numbers could be found in renaming `dot` to `dot_t` and having a `constexpr dot_t dot;` available, such that `* dot *` is valid.

Note that this pattern of absorbing a computation tree into the type of an object that is eventually decayed into the computed result is not novel; it is a somewhat common trick in building (e.g.) efficient fused operations and – probably, but who cares about 'em – autodiff pipelines. As such, this joke could almost certainly go deeper, but I leave that for future work.

Acknowledgments

Early and often.

Availability

Source code for this project is available via e-mail request to the first (and only) author. It is not available publically for fear someone might actually think this is a good idea.

References

[McCann, 2k22] McCann, J. (2k22). Grand challenges in programming languages position paper: What, if anything, does multiplication even mean? In *SIGBOVIK 2k22*. http://sigbovik.org/2022/proceedings.pdf.

Aesthetics

39 Real-Time Foliage Simulation

Emma Liu, Sanjay Salem, Daniel Zeng, Mia Tang, Hesper Yin, George Ralph, Anne He, Max Slater

Keywords: real-time, simulation, graphics, rendering

40 Using deep CNNs to prove that I look better than Tom Cruise and Shah Rukh Khan combined

Sagar Bharadwaj Kalasibail Seetharam

Keywords: CNN, Machine Learning, Image Compression

41 EBMP: Efficient Bitmap Encodings on Ethereum Virtual Machines

0XMOSTIMA, Yunsong Liu and Peiyuan Liao

Keywords: still image coding, blockchain, smart contracts, non fungible tokens

42 A Machine Learning Approach To Classifying Cuteness

Anoushka Shrivastava

Keywords: machine learning, classification, cuteness

43 Attractiveness Learning: A General Solution for the Cold-Start Problem

Yajuan Gu and Yuxun Lu

neural network, attractive mechanism, recommender system, cold-start problem

Real-Time Foliage Simulation

Emma Liu Sanjay Salem Daniel Zeng Anne He
Mia Tang Hesper Yin George Ralph Max Slater

Algorithm We explain our two-step process below:

1 Go outside.

2 Touch grass.

Using deep CNNs to prove that I look better than Tom Cruise and Shah Rukh Khan combined

Sagar Bharadwaj
Carnegie Mellon University

Abstract

Convolutional Neural Networks (CNN) have been used in the past to solve problems in many areas such as image classification, video analysis and drug discovery. However, no past work has considered using CNNs to prove that I look better than Tom Cruise (TC) and Shah Rukh Khan (SRK) combined. In this paper, I use a novel deep CNN architecture, Image2Float, and conduct surveys to prove conclusively that I indeed do look better than TC and SRK combined. 100% of the valid survey participants answer in the favour of the proof.

1 Introduction

Tom Cruise is a popular American actor and producer [5]. Shah Rukh Khan is an Indian actor, film producer, and television personality [4]. Popular public opinion considers these personalities attractive [2]. Refer to the Quora threads titled 'Do you think Shah Rukh Khan is handsome?' [1] and 'Why is Tom Cruise known as the most good-looking man on the planet?' [3] for detailed analyses. This implies, that when they are 'combined', the resultant personality will be drop dead gorgeous.

Image compression has been a long studied problem in computer science. Methods using deep neural networks for image compression have surpassed traditional codecs, achieving better compression ratio and reconstruction quality. In this paper, I train a deep CNN, Image2Float, with around 7 million parameters to compress 200 X 200 RGB images down to a single floating point number, realizing an unprecedented compression ratio of 120000. I urge the readers to ignore the fact that Image2Float was overfit on a set of 10 images, (5 TC + 5 SRK) to obtain these results. Refer to the Appendix for details on the model architecture, whose design, similar to most of the past work on Machine Learning, was a result of hope and randomness.

Figure 1 shows an outline of the main proof. The first line shows that SRK's image has been compressed down to a value of 0.5834 using the trained Image2Float network. TC's image has been compressed down to a value of -3.116. The combination (by summation) of the two encoded values is -2.538. Decoding the combined value using Image2Float yields the image shown in Figure 2.

I conducted an elaborate survey involving a total of 4 people to rank the image of the fictional personality, Cruise Khan (CK), in Figure 2 and my own image on an attractiveness scale from 0 to 10. My image used in the survey has not been included in the paper to respect the double blind nature of this prestigious conference. Unfortunately, 3 out of the 4 survey participants ranked my image below CK's stating, "There is no face in the world, real or fictional, that can be uglier than yours". I, the author, concluded that these responses are clearly biased against me and therefore decided to drop them from the final survey. The only unbiased remaining participant, me, agree that my own image ranks above Figure 2. This implies that 100% of the valid survey results agree that I look better than TC and SRK combined, thereby concluding the proof.

2 Appendix

Model Architecture: Figure 3 shows the layers in Image2Float, along with the number of parameters in each layer. I (mis)used 2 Nvidia RTX 2080 Ti cards to train Image2Float for this novel proof.

Code: As per tradition followed in the field, I have uploaded my undocumented and unclean code to GitHub (https://github.com/SagarB-97/Image2Float). I would like to confirm that the GitHub repository exists just so the paper can get past the reviewers who seek for an open source repository. It is practically unusable by the readers unless they have acquired the patience to struggle with it and set it up, which is only attainable after years of meditation in the Himalayas.

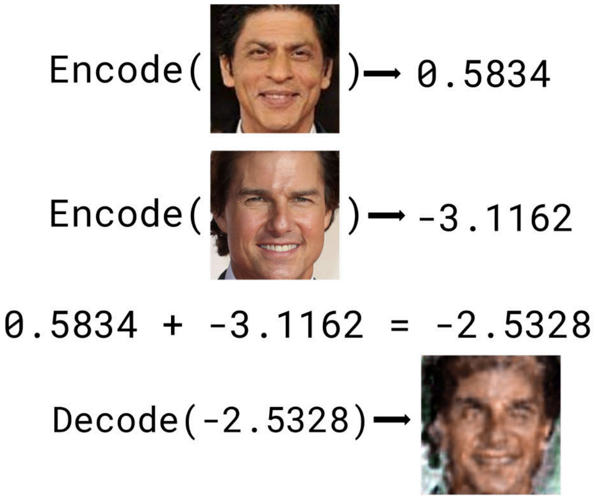


Figure 1: Outline of the proof.

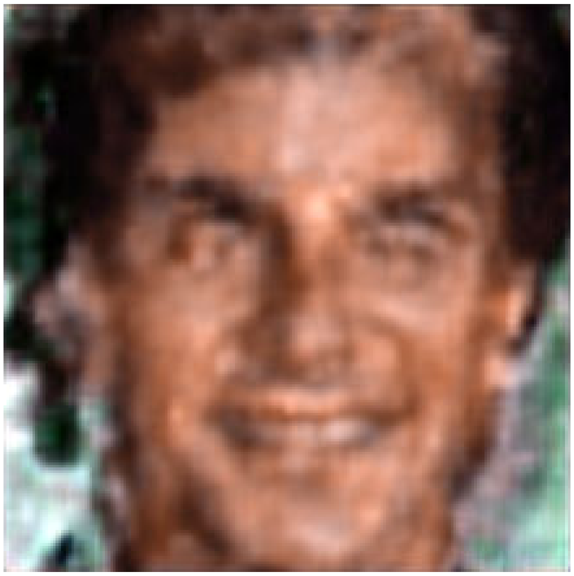
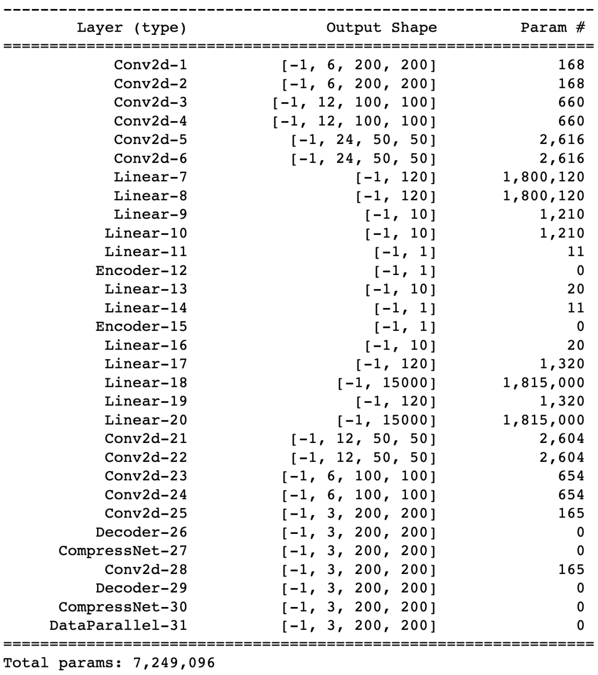


Figure 2: SRK and TC combined.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 6, 200, 200]             168
            Conv2d-2          [-1, 6, 200, 200]             168
            Conv2d-3         [-1, 12, 100, 100]             660
            Conv2d-4         [-1, 12, 100, 100]             660
            Conv2d-5           [-1, 24, 50, 50]           2,616
            Conv2d-6           [-1, 24, 50, 50]           2,616
            Linear-7                  [-1, 120]       1,800,120
            Linear-8                  [-1, 120]       1,800,120
            Linear-9                   [-1, 10]           1,210
           Linear-10                   [-1, 10]           1,210
           Linear-11                    [-1, 1]              11
          Encoder-12                    [-1, 1]               0
           Linear-13                   [-1, 10]              20
           Linear-14                    [-1, 1]              11
          Encoder-15                    [-1, 1]               0
           Linear-16                   [-1, 10]              20
           Linear-17                  [-1, 120]           1,320
           Linear-18                [-1, 15000]       1,815,000
           Linear-19                  [-1, 120]           1,320
           Linear-20                [-1, 15000]       1,815,000
           Conv2d-21           [-1, 12, 50, 50]           2,604
           Conv2d-22           [-1, 12, 50, 50]           2,604
           Conv2d-23          [-1, 6, 100, 100]             654
           Conv2d-24          [-1, 6, 100, 100]             654
           Conv2d-25          [-1, 3, 200, 200]             165
          Decoder-26          [-1, 3, 200, 200]               0
      CompressNet-27          [-1, 3, 200, 200]               0
           Conv2d-28          [-1, 3, 200, 200]             165
          Decoder-29          [-1, 3, 200, 200]               0
      CompressNet-30          [-1, 3, 200, 200]               0
     DataParallel-31          [-1, 3, 200, 200]               0
================================================================
Total params: 7,249,096
```

Figure 3: Model summary.

References

[1] Do you think shah rukh khan is handsome? https://www.quora.com/Do-you-think-Shah-Rukh-Khan-is-handsome. Accessed: 2022-03-31.

[2] Most handsome faces in the world. https://www.ibtimes.co.in/most-handsome-faces-world-tom-cruise-robert-pattinson-hrithik-roshan-others-top-list-check-705645: :text=Actor Accessed: 2022-03-31.

[3] Why is tom cruise known as the most good-looking man on the planet? https://www.quora.com/Why-is-Tom-Cruise-known-as-the-most-good-looking-man-on-the-planet. Accessed: 2022-03-31.

[4] WIKIPEDIA CONTRIBUTORS. Shah rukh khan — Wikipedia, the free encyclopedia, 2022. [Online; accessed 31-March-2022].

[5] WIKIPEDIA CONTRIBUTORS. Tom cruise — Wikipedia, the free encyclopedia, 2022. https://en.wikipedia.org/wiki/Tom_Cruise.

forgot to remove page numbers ach

EBMP: Efficient Bitmap Encodings on Ethereum Virtual Machines

0XMOSTIMA, Penguin Logistics, Leithania
YUNSONG LIU, Carnegie Mellon University, USA
PEIYUAN LIAO, hellsegga, ?

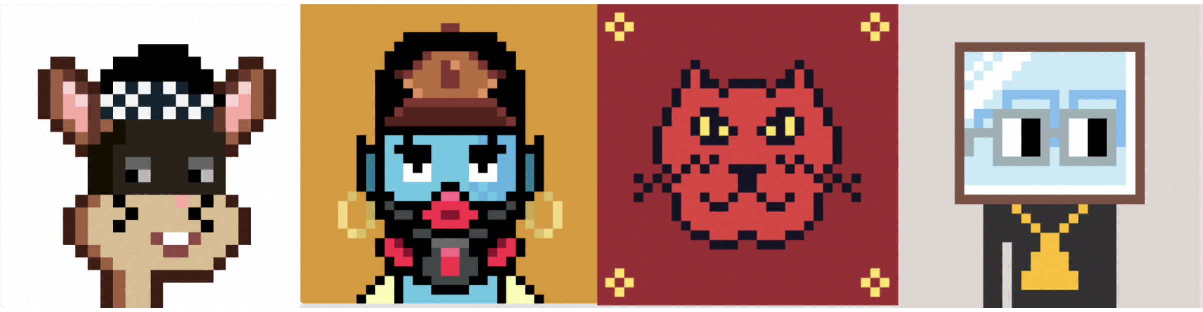


Fig. 1. Popular NFT Projects with On-chain Image: Anonymice [1], Chain Runners [2], Blitmap [3], Nouns [6]

The recent rising interests in Non-Fungible-Tokens (NFT) on cryptocurrency-backed blockchains have prompted a new series of efforts aiming to accurately store, read and render images on the Ethereum Virtual Machine (EVM, the software platform running on one of the leading blockchains, Ethereum), where smart contracts can enjoy "decentralized ownership and control" at the cost of increased computing spend. In this work, we present an efficient protocol to encode image data by directly constructing raw bytes for the device-independent bitmap (DIB) file format. The main function, implemented in Solidity, produces a shorter ERC-721-compatible tokenURI string when compared to existing methods while being more optimized in gas consumption.

CCS Concepts: • **Software and its engineering** → **Software libraries and repositories**; • **Computer systems organization** → *Peer-to-peer architectures*; • **Computing methodologies** → *Image compression*.

Additional Key Words and Phrases: still image coding, blockchain, smart contracts, non fungible tokens

ACH Reference Format:
0xmostima, Yunsong Liu, and Peiyuan Liao. 2022. EBMP: Efficient Bitmap Encodings on Ethereum Virtual Machines. In *SIGBOVIK '22: Special Interest Group on Harry Q. Bovik, April 8, 2022, Pittsburgh, PA.* ACH, 8 pages. http://sigbovik.org/

1 INTRODUCTION

Recently, there have been rising interests in crypto-currency-backed blockchains due to their potential technical and socio-economical impacts. One type of such systems, pioneered by Ethereum [13], presents itself as "a decentralized but singleton compute resource," where software developers can write applications, or "smart contracts," in domain-specific languages that are then compiled to bytecode for the virtual machines running on the "miners" maintaining the blockchain. For Ethereum, popular programming languages for smart contracts include Solidity [11], and Vyper [12], and the virtual machine is called the Ethereum Virtual Machine (EVM).

Ethereum's (reasonably) decentralized and immutable nature gives rise to a variety of use cases, one of which is non-fungible tokens, which are often associated with concepts of digital ownership, scarcity, and the creator

SIGBOVIK '22, April 8, 2022, Pittsburgh, PA

http://sigbovik.org/

economy. From a technical perspective, non-fungible tokens are implemented as an interface with a canonical application binary interface (ABI) so that any smart contract respecting it is expected to behave in a certain way (in this case, like a non-interchangeable unit of data). For the case of Ethereum, the standard specifying the behavior is called ERC-721 [5] with the method TOKENURI. It reads as follows:

```
/// @notice A distinct Uniform Resource Identifier (URI) for a given asset.
/// @dev Throws if `_tokenId` is not a valid NFT. URIs are defined in RFC
///  3986. The URI may point to a JSON file that conforms to the "ERC721
///  Metadata JSON Schema".
function tokenURI(uint256 _tokenId) external view returns (string);
```

With the "ERC721 Metadata JSON Schema" defined as follows:

```
{
    "title": "Asset Metadata",
    "type": "object",
    "properties": {
        "name": {
            "type": "string",
            "description": "Identifies the asset to which this NFT represents"
        },
        "description": {
            "type": "string",
            "description": "Describes the asset to which this NFT represents"
        },
        "image": {
            "type": "string",
            "description": "A URI pointing to a resource with mime type image/* representing
                the asset to which this NFT represents. Consider making any images at a width
                between 320 and 1080 pixels and aspect ratio between 1.91:1 and 4:5 inclusive.
                "
        }
    }
}
```

This implies that for an arbitrary client interacting with a smart contract on the EVM that respects the ERC-721 standard, they can expect to get a representation of the underlying data from a non-fungible-token by calling TOKENURI with the respective token ID.

The motivation of this paper came from the observation that many data represented by non-fungible tokens as of now is images due to its increasing adoption in digital art and speculative consumer brands. Currently, there are two ways such content is delivered: one is through traditional SaaS and IaaS providers, where TOKENURI points to the address of the content stored on the actual server, either a traditional cloud computing service provider or a storage-oriented blockchain. This can be seen as a compromise between centralization and decentralization: while the URL address string is immutable, the integrity and authenticity of the underlying content are not guaranteed by the consensus mechanism of the blockchains. The alternative approach, which is the subject of interest for this paper, is to store data directly on blockchain and efficiently render it using EVM itself. In other words, the returned string from TOKENURI would contain a well-formed ERC-721 metadata JSON without relying on any 3rd party rendering software or hosted servers. The main benefit of this approach is data availability and programmability: not only is the integrity of data guarded by the integrity of the blockchain itself, smart contracts running on EVM are also aware of such data, where subsequent actions like image transformations or zero-knowledge proofs on raw pixel arrays would also be possible. On the other hand, storing image data directly on-chain incurs an orders-of-magnitude extra cost for reading and writing large amounts of data to the blockchain.

EBMP is a new approach to this problem for non-fungible tokens running on Ethereum Virtual Machines: efficiently encode and render bitmaps for ERC-721-compatible smart contracts. While the majority of existing methods in the literature focus on drawing each pixel as a separate shape in HTML SVG graphics, we propose to instead directly construct raw bytes in the BMP file format [7], including the file header, device-independent bitmap (DIB) header and the image data, which is then encoded in base-64 and rendered first through the data:image/bmp;base64, mime type available for an <image> in SVG, then the data:image/svg+xml;base64, mime type available for the image field for a ERC-721 compatible tokenURI method. Our experiments demonstrate that EBMP produces a much shorter representation for a bitmap that is readable by common NFT platforms while achieving a considerable reduction in gas consumption compared to existing methods.

2 IMPLEMENTATION DETAILS

The full implementation of the encoding method is available below, with detailed in-line comments. EBMP accepts an arbitrary RGB image, with height and width divisible by 4 , pixels arranged in row-major order, and flattened with the three colors (1-byte each) adjacent to each other in red-green-blue order:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity 0.8.11;

import {Base64} from "./Base64.sol";

contract EBMP {
    function uint32ToLittleEndian(uint32 a) internal pure returns (uint32) {
        unchecked {
            uint32 b1 = (a >> 24) & 255;
            uint32 b2 = (a >> 16) & 255;
            uint32 b3 = (a >> 8) & 255;
            uint32 b4 = a & 255;
            return uint32(b1 | (b2 << 8) | (b3 << 16) | (b4 << 24));
        }
    }
    function encode(
        uint8[] memory image,
        uint32 width,
        uint32 height,
        uint32 channels
    ) public pure returns (string memory) {
        bytes memory BITMAPFILEHEADER =
            abi.encodePacked(
                string("BM"),
                uint32(
                    uint32ToLittleEndian(14 + 40 + width * height * channels)
                ), // the size of the BMP file in bytes
                uint16(0), // Reserved
                uint16(0), // Reserved
                uint32(uint32ToLittleEndian(14 + 40))
                // the offset, i.e. starting address, of the byte where the bitmap
                // image data (pixel array) can be found
            ); // total 2 + 4 + 2 + 2 + 4 = 14 bytes long
        bytes memory BITMAPINFO =
            abi.encodePacked(
                uint32(0x28000000), // the size of this header, in bytes (40)
                uint32(uint32ToLittleEndian(width)), // the bitmap width in pixels (signed integer)
                uint32(uint32ToLittleEndian(height)), // the bitmap height in pixels (signed integer)
                uint16(0x0100), // the number of color planes (must be 1)
                uint16(0x1800), // the number of bits per pixel
                uint32(0x00000000), // the compression method being used
                uint32(uint32ToLittleEndian(width * height * channels)), // the image size
                uint32(0xc30e0000), // the horizontal resolution of the image
                uint32(0xc30e0000), // the vertical resolution of the image
                uint32(0), // the number of colors in the palette, or 0 to default to 2^n
                uint32(0) // the number of important colors used, or 0 when every color is important
            ); // total 40 bytes long
        bytes memory data = new bytes(width * height * channels);
        // resharding
        for (uint256 r = 0; r < height; r++) {
            for (uint256 c = 0; c < width; c++) {
                for (uint256 color = 0; color < channels; color++) {
                    data[(r * width + c) * channels + (2 - color)] = bytes1(
                        image[((height - 1 - r) * width + c) * channels + color]
                    );
                }
            }
        }
        string memory encoded =
            Base64.encode(
                abi.encodePacked(
                    BITMAPFILEHEADER,
                    BITMAPINFO,
                    data
                )
            );
        return encoded;
    }
}
```

Base64.sol here is simply a default implementation for base 64 encoding, available in appendix.

To use EBMP in an ERC-721 compatible NFT contract, one may replace the image field from a traditionally URL pointing to the image to the base64 encoded string preceded by the MIME type header, like:

```
string memory json =
    Base64.encode(bytes(string(
        abi.encodePacked(
            '{"name": "EBMP", "description": "EBMP", "image":
            "data:image/svg+xml;base64,',
            Base64.encode(bytes(EBMP(img))),
            '"}'
        )
    )));
string memory ret =
    string(abi.encodePacked("data:application/json;base64,", json));
```

Where RET would be readable by common platforms supporting NFTs on Ethereum, like OpenSea [9].

3 ANALYSIS

We first present an analysis of popular on-chain image-encoding protocols to show that EBMP is the most flexible, then present gas and encoding-length analysis of EBMP (instantiated on 32x32 resolution, RGB) against one of the most permissible on-chain image encoding protocol, Pixelations [10].

Since BMP is one of the simplest image formats with a rather straightforward implementation requirement, Brotchain independently has developed a similar encoding method to EBMP to render bitmaps on the Ethereum blockchain [4]. However, some important differences remain:

(1) Brotchain adopts a palette-based encoding for BMP. In the regime where it saves the number of bytes needed to store an image on-chain, it would limit the number of possible colors presentable in an image to the palette used.
(2) Additionally, the bytes stored on blockchain do not exhibit spatial locality in the RGB color space, making further image manipulation harder.
(3) Finally, a prototype smart contract has been developed with a greyscale version of EBMP around the same time that Brotchain is released. Users may interact with it at rinkeby.0xyi.xyz.

Table 1 presents a brief and non-exhaustive overview of existing methods aiming to provide a general image encoding method on the Ethereum blockchain. Anonymice [1], and Chain Runners [2] are not included in this particular table due to the observation that the smart contract is not designed nor intended to encode images across a wide variety of domains, but instead focus solely on the very pattern on the non-fungible-token it seeks to support. Though similar is true for Nouns [6], the grouping color technique along with run-length encoding (RLE) does present an opportunity to be extended beyond encoding of the Nouns character; hence it is included in the table. We observe that EBMP offers a good balance between protocol freedom and data availability, where existing protocols either only supports a fixed number of colors or image resolution, or the data it provides is not always available (here, "Calldata" refers to the fact that the image data is only visible as input arguments to a function called on the blockchain, thus making it inaccessible for other functions running on the EVM).

Table 1. Comparison of Popular On-Chain Image Encoding Methods

Protocol	Type of data	Resolution	Number of bytes	Data availability	Color space
EBMP	Bitmap	Arbitrary (div. by 4)	$3wh$	Storage	RGB
Blitmap [3]	Bitmap	32x32	268	Storage	4 colors
Pixelations [10]	Bitmap	32x32	736	Storage	32 colors
Nouns [6]	Bitmap/Vector	32x32	RLE	Storage	Varied
Brotchain [4]	Bitmap	Arbitrary	$768 + wh$	Storage	256 colors
0xmons [14]	GIF	Arbitrary	up to 125 KB	Calldata	Arbitrary

Table 2 records the average encoding length and gas cost for random 32x32 RGB images on EVM across different protocols. We observe that when compared to existing methods that offer general-domain image encoding support off-the-shelf, EBMP provides a 2.8x reduction in terms of gas consumption and 1.4x reduction in encoding length even though using a far-richer color space with more bytes needed. As a separate analysis, we also include gas profiling results for Brotchain. We remark that although it produces both a shorter encoding string and a lower gas cost, EBMP is capable of representing up to 2^{24} colors in RGB format, which is $2^{16} = 65536$ times more than that of Brotchain. Additionally, since Brotchain does not exhibit spatial locality of data, resharding of raw pixel bytes is no longer needed, which will also contribute to a lower gas cost. We verify this by turning off resharding for EBMP as well and re-run the test, which indeed generated even lower gas cost and deployment cost.

Table 2. Gas and average encoding length of EBMP against Pixelations on 32x32 Resolution

Protocol	Gas	Encoding Length	Bytes	Deployment Cost	Spatial Locality	Colors
EBMP	4798684	4168	3072	551587	**Yes**	**2^{24}**
Pixelations	13390461	58575	**736**	1889790	**Yes**	2^5
Brotchain	437448	**2826**	1792	799231	No	2^8
EBMP (no resharding)	**289851**	4168	3072	**413049**	No	**2^{24}**

4 BROADER IMPACTS

It is still worthy of mentioning that although EBMP has achieved significant cost reduction against existing methods, the cost is still orders of magnitude larger when compared to traditional IaaS and SaaS service providers. Table 3 presents such an analysis, where we derive an upper bound for platforms running EVM emulators (instead of a blockchain) by assuming that a single EBMP call finishes in 2 seconds (which is true for all platforms compared).

Table 3. Analysis of EBMP against computing platform

Platform	Cost	Uptime	Data Integrity	Censorship Resistance
Ethereum [1]	\$1904.75	1 (1.07 PH/s POW) [2]	Always (POW)	If set up correctly [3]
Avalanche (EVM compat.) [4]	\$35.06	1 (validators' POS)	Always (POS)	If set up correctly
GCS (n2-highmem-8) [5]	$< \$0.0002912$	likely $\geq 99.5\%$ [6]	SLA	If they want to
AWS (t4g.2xlarge) [7]	$< \$0.0001494$	likely $\geq 99.99\%$ [8]	SLA	If they want to
Paperspace C6 [9]	$< \$0.00008889$	likely $\geq 99.99\%$ [10]	SLA	If they want to
my M1 MacBook Pro [11]	$< \$0.0001268$	me	also me	N/A

[1] at the time of writing, Ethereum's price is \$2832.87

[2] POW stands for Proof-of-Work, and POS stands for Poof-of-Stake, which are different forms of achieving consensus on blockchains. Ethereum and Avalanche guarantee full uptime and data integrity given that consensus is achieved, which is theoretically breakable under a series of scenarios that are outside the scope of this paper.

[3] If the set-up to run EBMP on blockchains is incorrect, then it may not be censorship resistant.

[4] at the time of writing, Avalanche's price is \$80.44

[5] on-demand hourly price for this instance at the time of writing is \$0.52405

[6] https://cloud.google.com/compute/sla

[7] on-demand hourly price for this instance at the time of writing is \$0.26880

[8] https://aws.amazon.com/legal/service-level-agreements/

[9] on-demand hourly price for this instance at the time of writing is \$0.16

[10] https://www.paperspace.com/security

[11] assuming a laptop lasts for a year, and the cost of purchasing a M1 MacBook Pro is \$1999.00.

5 CONCLUSION

EBMP is a gas-efficient way to encode bitmaps on Ethereum Virtual Machines, which has implications in non-fungible-token implementations running on popular blockchains. Future improvements could be made on different resharding methods and extending the protocol to allow encoding of images with length or width not divisible by 4.

ACKNOWLEDGMENTS

We thank Diana from A-SOUL for the emotional support. You should definitely check out some of their wonderfully-made music videos:
https://www.bilibili.com/video/BV1vQ4y1Z7C2
https://www.bilibili.com/video/BV1FX4y1g7u8/

REFERENCES

[1] Anonymice. 2021. *Anonymice.* https://opensea.io/collection/anonymice
[2] Mega City. 2021. *Chain Runners.* https://opensea.io/collection/chain-runners-nft
[3] dhof. 2021. *Blitmap.* https://opensea.io/collection/blitmap
[4] divergence. 2021. *Brotchain.* https://opensea.io/collection/brotchain
[5] W. Entriken, D. Shirley, J. Evans, and N. Sachs. 2018. EIP-721: Non-Fungible Token Standard. *Ethereum Improvement Proposals* no. 721 (Jan. 2018). https://eips.ethereum.org/EIPS/eip-721
[6] Nouns Foundation. 2021. *Nouns.* https://opensea.io/collection/nouns
[7] Ron Gery. 1992. DIBs and Their Use. https://docs.microsoft.com/en-us/previous-versions/ms969901(v=msdn.10)?redirectedfrom=MSDN
[8] Georgios Konstantopoulos. 2021. *foundry.* https://github.com/gakonst/foundry
[9] OpenSea. 2022. OpenSea, the largest NFT Marketplace. https://opensea.io/
[10] Pixelations.xyz. 2021. *Pixelations.* https://opensea.io/collection/pixelations-xyz
[11] Solidity Team. 2022. *Solidity.* Ethereum Foundation. https://docs.soliditylang.org/en/v0.8.13/
[12] Vyper Team. 2022. *Vyper.* https://vyper.readthedocs.io/en/stable/
[13] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.
[14] xmon.eth. 2021. *0xmons.* https://opensea.io/collection/0xmons-xyz

A SOFTWARE ARTIFACTS

The software artifacts to reproduce the experiments in the paper, including an additional copy of the EBMP protocol, can be found here: https://github.com/0xmostima/EBMP. It is implemented in Solidity with Foundry [8].

B SAMPLE BASE64 IMPLEMENTATION IN SOLIDITY

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

/// [MIT License]
/// @title Base64
/// @notice Provides a function for encoding some bytes in base64
/// @author Brecht Devos <brecht@loopring.org>
library Base64 {
    bytes internal constant TABLE =
        "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

    /// @notice Encodes some bytes to the base64 representation
    function encode(bytes memory data) internal pure returns (string memory) {
        uint256 len = data.length;
        if (len == 0) return "";
        // multiply by 4/3 rounded up
        uint256 encodedLen = 4 * ((len + 2) / 3);
        // Add some extra buffer at the end
        bytes memory result = new bytes(encodedLen + 32);
        bytes memory table = TABLE;
        assembly {
            let tablePtr := add(table, 1)
            let resultPtr := add(result, 32)
            for {
                let i := 0
            } lt(i, len) {

            } {
                i := add(i, 3)
                let input := and(mload(add(data, i)), 0xffffff)
                let out := mload(add(tablePtr, and(shr(18, input), 0x3F)))
                out := shl(8, out)
                out := add(
                    out,
                    and(mload(add(tablePtr, and(shr(12, input), 0x3F))), 0xFF)
                )
                out := shl(8, out)
                out := add(
                    out,
                    and(mload(add(tablePtr, and(shr(6, input), 0x3F))), 0xFF)
                )
                out := shl(8, out)
                out := add(
                    out,
                    and(mload(add(tablePtr, and(input, 0x3F))), 0xFF)
                )
                out := shl(224, out)
                mstore(resultPtr, out)
                resultPtr := add(resultPtr, 4)
            }
            switch mod(len, 3)
                case 1 {
                    mstore(sub(resultPtr, 2), shl(240, 0x3d3d))
                }
                case 2 {
                    mstore(sub(resultPtr, 1), shl(248, 0x3d))
                }
            mstore(result, encodedLen)
        }
        return string(result);
    }
}
```

A Machine Learning Approach To Classifying Cuteness

Anoushka Shrivastava
Carnegie Mellon University

March 25, 2021

Abstract. Computer science students have conquered many problems, but finding their ideal match is not one of them (and unfortunately algorithmic solutions tend to be lacking in this area). Nevertheless, in preparation for a time (in the far, far future) that we get stormed with hundreds of dating offers instead of Gradescope receipts in our inboxes, this paper aims to devise a machine learning model to filter through our options by classifying whether people are cute or not. This classification is life-saving since by narrowing down candidates for them, we save CS students time to focus on their personal hygiene, a topic on which an entire study of its own can (and probably should) be conducted.

I. Introduction.

Anyone intrigued by the abstract is probably a lost CS student, so instead of scaring (or scarring) you with statistics about the percentage of students who are able to find love (hint: think less than the probability of rolling a 7 on a 6-sided die), I will use the introduction as encouragement for you. If you're at the point where you need machine learning to solve your problems, your desperation probably leads you to have low standards for a partner, so you will find someone eventually. However, this is also why the model is important: since your bar is so low, when you finally do get offers, we can be sure to use the model to reject the people that you should be immediately rejecting under normal circumstances.

II. Methods.

When I finally got an idea for this paper a few hours before the deadline, the only people around at home were my mother, father, and sister. Therefore, my dataset consists of these three, and my own photos. I made many attempts to find others to take pictures of, including following my neighbors on their walks, but I was informed by their lawyer that the ethics of this approach were a bit controversial.

In an effort to produce more training data, I decided to include front, side, back, up, down, upside-down, left, right, north, south, east, west, and diagonal profiles of each family member. I also included them sitting, standing, jumping, kicking, swinging, eating, and any other pose corresponding to my first-grade vocabulary verb list of the week of September 27th. Finally, because the dataset was still not full enough, I included pictures of household objects such as my sofas and air conditioner.

I began with the intent to lead with a supervised training approach. However, after my mother learned that I had labeled the worm in our garden cuter than her, she suggested that if I wanted to keep my tuition at CMU, I best make this an unsupervised learning model. I obliged, especially since labeling my own image would probably be a conflict of interest.

Honestly, after rereading my methods on data collection, I'm not quite sure that the rest of my methods are so sound. In fact, I'll just keep the rest of this section to myself.

III. Results and Conclusions.

So, it turns out the cutest person is the kitchen spoon in our left drawer. While I am surprised that the spoon beats even my flattering diagonal profile, I still think the model's confidence of 1003.27% seems legit and makes it trustworthy.

IV. Future Work.

After my family realized that they had lost to a spoon, they informed me that they were drafting up a 10-year research project titled "How long can one college student survive on dining hall food?" with me as the test subject. With sincerely no offense to my university, I do not wish to participate in this study, and will be terminating my work on this project.

V. Acknowledgements

I would like to thank my friends Keevyu and Karen for unintentionally inspiring this project. I would also like to thank Sheryl Mathew and Preethi Krishnamoorthy for providing feedback and proofreading. Finally, I would like to thank my coffee mug, who was disappointed that the spoon was cuter than it (mug, you'll always have a special place in my heart).

VI. References

None.

Attractiveness Learning: A General Solution for the Cold-Start Problem

Yajuan Gu
Extreme Advanced Technology Department
A Super, Super, Super Tiny Start-up Company
Somewhere, Southern Part, China
heng6534605@163.com

Yuxun Lu[*]
National Institute of Informatics
University for Advanced Studies
Chiyoda, Tokyo, Japan
lu-yuxun@nii.ac.jp

Abstract

The cold-start problem is an essential issue in the recommender system. Recommender systems based on Collaborative Filtering or Deep Neural Network need sufficient user-item interactions to optimize the parameters. It is difficult to provide recommendations related to new users and new items because of the insufficient records. We proposed an attractiveness Learning approach as a general solution for the cold-start problem. Our approach does not need any data, GPUs, hard-working researchers, graduate school students (especially Ph. D. students). Experiments show that our method has achieved the state-of-the-art performance on zero-shot recommendation for items and users.

1 Introduction

Recommender system has become an core component in wide categories of online services. It encourages users to keep surfing so that the online platform can take the money out from pockets of the users, bigly and silently. Recommender systems suffer from the cold-start problem, i.e. they cannot provide satisfying recommendation to new items and new users because of the insufficient interactions (Lee et al., 2019).

Current methods alleviate the cold-start problem by interaction data from existing users and items to help the fast adaption of the parameters in the recommender system in a meta-learning framework (Lee et al., 2019; Finn et al., 2017; Wang et al., 2021; Snell et al., 2017; Sankar et al., 2021). These solutions are with a common flaw: all of them require hard efforts from diligent researchers and Ph. D. students. The inner loop in MAML (Finn et al., 2017) training procedure results in unbearably slow optimization, and the usage of GPU servers accelerates the carbon emission indirectly (Strubell et al., 2019).

We proposed a novel solution for the cold-start problem in zero-shot learning. Our approach employs attractive persons to recommend items to consumers. Our contributions are summarized as follows.

1. Our approach does not need any electric computation resources and outperform all methods that need them in the zero-shot scenario for cold-start problem.

2. The carbon emission of our method is significantly reduced compared to methods that need computational resources.

2 Methodology

The key point in our method is "how to find attractive persons and persuade them to do the sale". Both authors realized such research must satisfy the standard of experiments that related to human. However, the regulations and rules, although absolutely vital, are long and tedious for laymen. Both authors (Gu and Lu) have read the regulations and rules about, you know, about, ugh, 5 minutes-ish and found out that the management is less strict if the human involved in the experiments are limited to the authors *per-se*, so the persons involved in our methods are the authors of this paper, Yajuan Gu and Yuxun Lu.

According to their common friends, Gu "is charming and outgoing.", and the comment for the same question in regard to personality about Lu is "I'd rather not to say for the sake of our friendship". We use Lu as the control group and Gu as the experimental group. Our method is described in Algorithm 3.

3 Experiments

We conduct the experiments in (1) a restaurant use a red hair person in cartoon style as its logo; (2) a

Algorithm 1: the Attractiveness Learning Framework

```
Data: A group of consumers $\mathcal{C}$. A set of all
      items in store $\mathcal{G}$. The seller $s$.
Sort $g \in \mathcal{G}$ by the profit as ordered set $\mathcal{G}^*$.
Estimate the gullibility of all $c \in \mathcal{C}$ by the $s$.
while $\mathcal{C}$ is not empty do
    Greeting $c^*$ with the highest gullibility.
    while $c^*$ is not leaving do
        Try to sell $g^* \in \mathcal{G}^*$ to $c^*$.
```

luxury shop whose design is famous for being similar to checkerboard; (3) A membership mall being known for its "suspicious" low price. We take binary Normalized Discounted Cumulative Gain and Ratio in Total Sales as performance measures. The Ratio in Total Sales is

$$r = p(s)/p^*. \tag{1}$$

$p(s)$ is the sale amount from s in Algorithm , and p^* is the total sale amount of the day. The equation for binary NDCG is (relatively) too complicated for the lazy authors to type here. Please use the search engine Google [1] for the formula. The results for 6 days sale in 3 domains are shown in Figure 1 and Figure 2, respectively.

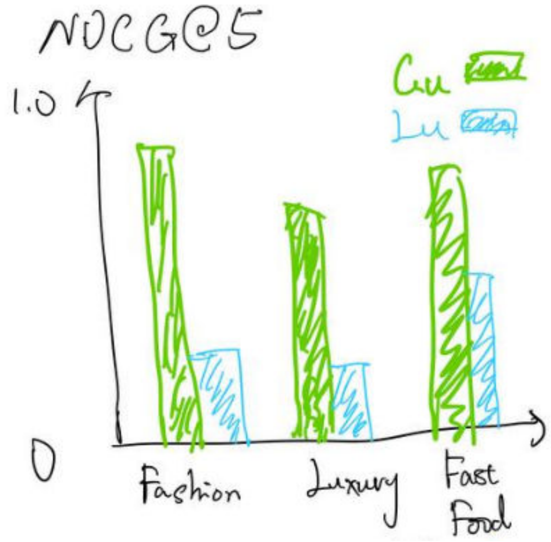


Figure 1: The averaged NDCG@5 for Gu and Lu in 3 domains. Drawn by Yajuan Gu (the first author).

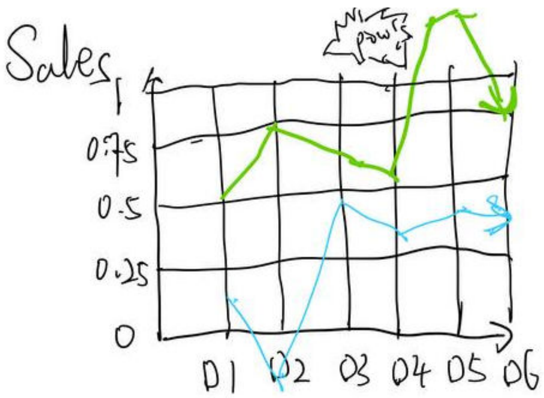


Figure 2: The Ratio in Total Sale for Gu and Lu in the "Fast Food" domain for 6 days. Drawn by Yajuan Gu (the first author).

3.1 Discussion

The results in Figure 1 and Figure 2, show that the attractiveness is a critical property for the saler s in Algorithm 3. Especially, the total sale ratio for the "Fast Good" domain on Day 5 is over 1.0 for Gu because a generous customer paid a big tip after ordering [2] while Lu arrived negative ratio in Day 2 because he was caught by the cantine manager when sneaking burger and fries in the kitchen and has to pay the fine.

It worth note that the saler s has no information about the demographic information of the consumers in $\mathcal{C}$. Hence, every consumer is treated as a new consumer in the system. Beside, in Algorithm 3, the saler s does not care about the attributes of the goods in $\mathcal{G}$ at all. Therefore, it is unimportant if the item is new or not.

4 Related Work

Attractiveness based recommender system has been known in many popular cultures. For example, the character "Penny" in "the Big Bang Theory" and "Max" in the "2 Broke Girls". But it seems that no rigorous research has been done yet for investigating the impact of attractiveness in the recommender system domain. We wish this letter will encourage more researchers to extend the future investigation about this topic.

5 Conclusion

We proposed a attractiveness learning framework in this letter. The system does not rely on any

[1] https://www.google.com

[2] Both nations in which the authors resident do not have tipping tradition.

data to solve the cold-start problem in user side and item side. The experiment result shows that attractiveness learning can significantly improve the profit resulted from recommendation.

Acknowledgement

We appreciate the suggestions from all reviwers, if any, about the spelling and grammartic issues in this paper. Especially the opinion that suggested an extremely expensive service on editing. Our answer, as always, is "hack know. VR nut going two yuzit."

References

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1126–1135.

Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. 2019. Melu: Meta-learned user preference estimator for cold-start recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1073–1082.

Aravind Sankar, Junting Wang, Adit Krishnan, and Hari Sundaram. 2021. Protocf: Prototypical collaborative filtering for few-shot recommendation. In *Fifteenth ACM Conference on Recommender Systems*, pages 166–175.

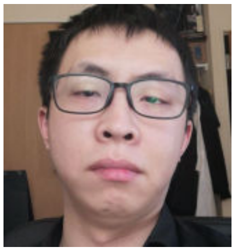
Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 4080–4090.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Li Wang, Binbin Jin, Zhenya Huang, Hongke Zhao, Defu Lian, Qi Liu, and Enhong Chen. 2021. Preference-adaptive meta-learning for cold-start recommendation. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pages 1607–1614.

A Author Biography

Gu Yajuan is a senior researcher in a super start up company. Gu is also the saler, door keeper, cook, cheif, account, treasory, general affair manager, human resource manager, stakeholder, general manager of the company. The research interest Gu is about all aspects in fashion and food.



Lu Yuxun is a slob in the National Institute of Informatics in Japan. Lu focus on the research of rushing paper draft before submission deadline.

Hardware

44 Correct-It-Yourself Paper Updater

Thomas Chick

Keywords: corrections, arts and crafts, reader participation, sacrilege? sure

45 Harder Drive: Hard drives we didn't want or need

Tom Murphy

Keywords: hard drive, block device, ipv4, tetris, covid-19, chainsaws

46 Just a Regular Paper

Jason Wang

Keywords: Money, Finance, Banking, Social Issues, Machine Learning, Ohio

47 Wearable RF-shield repositories

Abe Wits and J. Mijn

Keywords: security, wifi, cable

Thomas Chick's

CORRECT-IT-YOURSELF PAPER UPDATER

Now with 100% fewer errors the author noticed after going to press!
Includes new 2021 figures!
Improved contrast in black and white!

Simply:

1. Take your copy of ""“The SIGBOVIK paper to end all SIGBOVIK papers” will not be appearing at this conference" from *SIGBOVIK 2021*.
2. Cut out all the tables and figures from this page and paste them into the correct positions.
3. Enjoy your newly updated paper!
4. ~~Feel immense embarrassment.~~

It's really that easy!

	‘07	‘08	‘09	‘10	‘11	‘12	‘13	‘14	‘15	‘16	‘17	‘18	‘19	‘20	‘21
Papers	35	25	39	39	20	25	25	24	17	23	35	30	45	39	43
(to date)	35	60	99	138	158	183	208	232	249	272	307	337	382	421	464
Authors	33	33	43	78	37	31	34	27	19	72	48	56	60	66	65
(to date)	33	58	91	154	182	203	230	250	259	319	356	396	444	492	541

Table 1: Numbers of papers and authors at each conference, individually and cumulatively

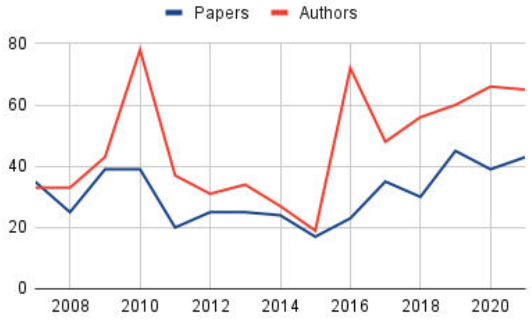


Figure 1: Numbers of papers and authors at each conference, individually

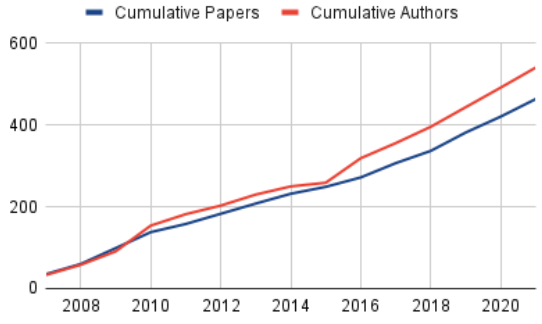


Figure 2: Numbers of papers and authors at each conference, cumulatively

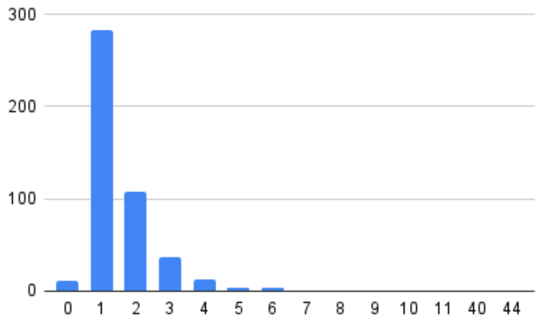


Figure 4: Number of papers per number of authors

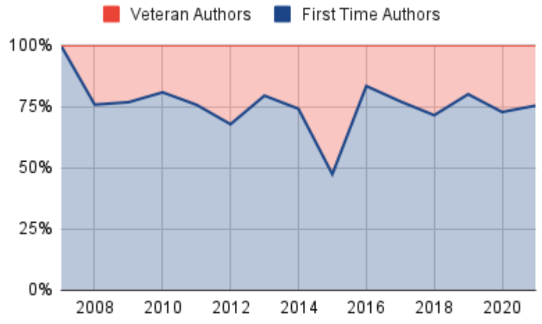


Figure 3: Proportion of previous authors to first-time authors at each conference

Harder Drive: Hard drives we didn't want or need

Dr. Tom Murphy VII Ph.D.

7 April 2022

1 Introduction

It's currently the longest 2020 ever on record, seemingly with a new annoying or demoralizing twist every week. To me, the most effective distraction is making things, but what to make? And what if it's no better than what was made before? Then perhaps you will experience an annoying or demoralizing twist. So, here's an approach that's so robust I don't even mind giving it away: Make something that nobody would want, or need. The competitive landscape for needless things is relatively uncrowded, for obvious reasons. A good way to think of things that we might not want is to consider common abstractions that have many instantiations (e.g. chess [17], boolean logic [18], integer math [11]), and come up with new ones. Moreover, judge them according to nontraditional criteria. This way, they will "be better"—in at least some sense—than what was made before. This is the secret of SIGBOVIK.

This paper investigates several storage devices ("Harder Drives") that we didn't want, or need.[1] In doing so, we will find inspiration from some vexing current events. Even if only tangentially related, creating structured thoughts on the periphery may help us digest them. It works for me. This is the laxative of SIGBOVIK.

Despite setting out to do an "easy, fun project," of course I managed to make each one much more difficult than I initially anticipated. This is the curse of SIGBOVIK. Bon appétit.

1.1 Chainsaws

At first glance, it seems that the maximum number of chainsaws that a person can wield is two: One per hand. This is known as *dual-wield*. It may be possible to achieve more using "one-man-band" arrangements (shin- and knee-wielded chainsaws, elbow saws, a mouth-held chainsaw activated by blowing into it like a harmonica) but a more natural way to scale is by juggling the saws into the air. Now at second blush it seems that an arbitrary number of saws can be simultaneously equipped, by simply throwing the chainsaws higher into the air. This configuration is known as ∞-wield. However, throwing chainsaws higher and higher eventually reaches physical limits. Once the thrown saws reach the escape velocity of $11.186^{km}/_{s}$, they will not fall back to Earth, and can hardly be considered brandished by the juggler. Just shy of this, they reach some maximum height before returning. This loop has two problems: First, it is not the longest possible airtime. Second, it can only be used for one chainsaw at a time, as the saws will otherwise interfere with one another along the out-and-back path.

Upon a third look, we need not throw the chainsaws straight upward; instead we can juggle the saws into orbit around Earth. Ignoring air resistance (the chainsaws cut through air like butter) and assuming double precision floating point with discrete time steps of 0.1 seconds will suffice (probably not), my simulation[2] finds the longest orbit that returns the chainsaw to 1.5m above the surface is 335.36 hours.

There are many such orbits to fill with chainsaws, but the limiting factor seems to be the density of chainsaws near the wielder (where the many orbits would interfere). Assuming each orbit is basically parallel here, and consists of chainsaws about 0.2 m wide (they can be efficiently packed in "69 configuration") moving at the maximum velocity of 11,186m/sec, we see 55,930 per second. We could probably fit about three deep and three high on each side of the body, for 18 times that. So we have 1 million chainsaws per second, for 335.36 hours, which is 1.215×10^{12}, a configuration known as tera-wield. This requires expert juggling skills.

Why am I talking about this? Overall, the important lesson here is creativity with dimensional analysis: We can achieve a quantity of *chainsaws* by multiplying some *chainsaws per second* by some *seconds.*

1.2 Juggling with data

Now imagine that instead of chainsaws, we are juggling something more dangerous: Data.

One potential setup would be a powerful directional antenna, which broadcasts a stream of data towards the horizon. For radio waves below 40 MHz, significant reflections off the ionosphere occur, bouncing the waves back to Earth.

*Copyright © 2022 the Regents of the Wikiplia Foundation. Appears in SIGBOVIK 2022 with the fatal Input/Output error of the Association for Computational Heresy; *IEEEEEE!* press, Verlag-Verlag volume no. 0x40-2A. 6.6e-11 Nm^2kg^{-2}

[1] Source code can be found at `sourceforge.net/p/tom7misc/svn/HEAD/tree/trunk/pingu/`.

[2] Someone equipped with Johann Sebastian Kepler's laws could just solve this exactly.

They may also reflect off the ground, and again off the ionosphere, and in principle make their way fully around the planet. The antenna is paired with a receiver in the same location, which accepts the signal and retransmits it, "juggling" it back into circulation.

Since a full trip around the Earth is 40,000 km (and significantly more in this setup due to reflections) and radio waves take time to propagate, this orbit takes at least 150ms to complete. As a result, we can have 0.15 sec × 40,000,000 bits/sec = 750 kilobytes of data outstanding in the steady state. When data we are interested reaches the receiver we can "read" it, and of course we can choose to retransmit modified data to perform a "write," This is similar to the rotation of a hard drive, with the fixed read/write head waiting for the "orbiting" platter.

This author does not have sufficient skill to construct such a system, which would probably not work in practice anyway. The reflections probably do not make it all the way around the Earth, and the noise from other radio waves would offer significant interference.

A superior arrangement would place repeater towers along the great circle. This would clearly work but would require an investment in real estate throughout the world. But the main reason not to do this is that 750kb is a trivial amount of storage; a similar magnitude is found incidentally in disposable consumer devices (Section 4).

Of course, if we are retransmitting the signal anyway, we do not need to send it in the same direction. It is much simpler to send it directly back, like an echo.

1.3 ICMP Echo

Sorry to remind you about the Internet, which seems to be making the whole world collectively dumber and meaner, but this section concerns a hard drive made from the transmission delays of the network itself.

Back when the internet was a collaborative and basically nice place, the Internet Control Message Protocol (ICMP) was proposed as a way for network nodes to help each other out. This protocol allows sending messages like "hey this address is down!" or other tips. Since it is easy to forge ICMP messages, most off these have potential for abuse (like telling you the site you're talking to is down) and are no longer commonly honored. (As another indicator of the era, these internet standards are known as "Requests for Comment", with ICMP described in RFC 792 [21]. On the modern internet we still have requests for Comment, but they are almost universally accompanied by requests for Like and Subscribe.)

However, many hosts will still respond to an `ECHO` packet with `ECHO REPLY`. This is typically used to "ping" a host: The source sends `ECHO` to the destination with some identifying information and an embedded timestamp; the destination sends `ECHO REPLY` with that same data back to the source, and the source can calculate the round-trip time.

Since there are hosts throughout the world that will already reply to `ECHO` messages, this could be a perfect setup for juggling data! The `data` field of the `ECHO` can store the bytes of interest. When we receive an `ECHO REPLY` we will "read" or "write" that data if needed, and then immediately broadcast another `ECHO`. Since we do not retain the data otherwise, it will be stored "inside" the internet itself: Inside the buffers of routers but also as moving photons inside fiber optics, flowing charge in ethernet cables, and so on.

In principle we should be able to saturate our internet connection with outgoing `ECHO` and incoming `ECHO REPLY`; even on a consumer plan (these days on the order of 1 gigabit/sec) we may be able to store significant amounts of data. If we ping a host on the Earth's antipode, the round trip time will be at least 150ms (speed of light and circumference are limits here as well). $1Gb/sec \times 0.15sec$ = 833 Megabytes.

In practice this proves to be much more difficult. Alas, even the apparently harmless `ECHO` has been regularly abused for denial-of-service attacks, such as the "Ping of Death" [27] and "Smurf attack" [28]. Thus, hosts almost always have hard limits that we have to work within. We will face the following difficulties that cause us to fall far short of the ideal above:

1. Hosts limit the size of an `ECHO` packet they will respond to. 512 bytes of payload is a typical limit for a fairly permissive host,[3] but many will reject payloads more than a few dozen bytes. The IP header (20 bytes) and ICMP header (8 bytes) thus contribute significant overhead.

2. Hosts have global limits on the rate of incoming and outgoing ICMP messages.

3. Consumer internet connections have built-in throttling of ICMP messages, perhaps to limit the impact of Denial-of-Service attacks originating from their networks.

4. Pings are "best effort" and readily dropped by congested routers without retries (this can even be a desirable property for measuring network congestion).

1.4 Pinging the internet

While developing code that can process many thousands of pings per second and investigating these limitations, I figured I might as well ping the entire internet.

Here by internet I mean "IPv4 address space." I don't care about IPv6 which has *way too many addresses* (plus like, call me when you are at least version 7, right?). There are only 2^{32} IPv4 addresses, which is no longer that big of a number. I wrote a fairly simple program `pingy.exe` which pings all of the hosts of the form `*.*.c.*` for some $c \in \{0, \ldots, 255\}$ in a random order. For each one it saves

[3] Allegedly, "all hosts are required to be able to reassemble datagrams of size up to 576 bytes,"[22] but I guess most do not care about this or consider it better than dropping all pings. There are not many legitimate uses for a payload of this size, anyway.

the number of milliseconds of round-trip time (or records special sentinel values for “timeout” or “wrong data returned”) in a single byte. This results in 256 files, which assembled are 4.2 Gigabytes.

This turned out to be much more logistically challenging than I expected. Naïvely I should be able to send millions of pings per second, but the packets are dropped somewhere if I exceed about 1000 pings/sec. Even at 1000 pings/sec (a trivial amount of bandwidth) this behavior seems to wreak havoc on my home network; other devices sharing the connection get extremely bad performance, a no-no for the Work-from-Home video call lifestyle of the pandemic. This could be because my internet provider throttles ICMP; it could also be that some hardware or software in the path is not able to handle thousands of different IP addresses each second (e.g. there may be fixed-size NAT tables). I tried using a VPN, but this had a much lower success rate; the VPN egress point probably throttles ICMP to prevent DDoS attacks, and it’s possible that many internet gateways also simply block ICMP from known VPN endpoints since they are obvious choices for people up to no good. Anyway, what I thought might take a few hours or a weekend ended up taking months. Eventually I rented time on several machines in different data centers to parallelize the process; this also produced a higher ping response rate than my home network, so I redid all of the already-completed sections for uniformity. The results make a nice poster, though, and are in Figure 1.

9.18% of addresses responded successfully within 4 seconds. Only 4,529 hosts (0.000105%) replied with the wrong data.

2 Harder Drive: Pingu

At last, I yearn to build a virtual hard drive using the ideas above. It will be called `pingu`.[4] To do that, I first had to figure out how to make a hard drive. This is no big deal.

2.0.1 nbdkit

In UNIX, storage systems are abstracted as “block devices.” Like all things in UNIX, it is conceptually “just a file,” but then gets complicated with all sorts of concessions for efficiency. Fortunately, efficiency is a non-goal for this project. We could implement these drives as kernel modules that implement the basic operations of a block device. This would be a bad choice because of the number of userspace facilities we want to use, and also because I would have to do a *lot* of rebooting as my myriad bugs panicked the kernel.

`nbdkit` (for “no big deal” kit) is a library for creating and mounting Network Block Devices in userspace. Network Block Device (for “NBD”) is a protocol for communicating with a quote-unquote block device (for example, a physical hard drive, or a virtual drive like a file containing a DVD-ROM image, or a block of memory, or the variety of weird drives considered in this paper) over a network. It’s also straightforward to use with a local quote-unquote network (i.e. UNIX domain socket).

In order to create a device, you implement functions like `pread` (read some bytes from a region in the device and copy them into the caller’s buffer) and `pwrite` (same in reverse). There are also many optional operations (e.g. “fast zero” a region) for efficiency, plus hints for `nbdkit` or the kernel to know how to optimize data layout on the drive. For example I was charmed to see a flag `is_rotational` that describes whether the drive is based on spinning platters, which presumably is used as a hint that sequential reads/writes are more efficient. The block device is compiled as a shared object that can be loaded into `nbdkit`’s server, then attached (as root) as a block device like `/dev/nbd0`. At this point, the device can be formatted with some filesystem.

2.0.2 Implementation

The smallest drive that can be formatted and mounted on a normal Linux machine is 51,200 bytes, using the FAT12 filesystem common on DOS floppy disks in the 1980s [26].[5] So the device consists of one hundred 512-byte blocks. Each block will be stored inside multiple outstanding pings (for redundancy) with a 512-byte payload.

Despite pinging the whole internet in Section 1.4, we use a fixed set of IP addresses here. The reason for this is that we want a set of IP addresses that are stable, reliable, and have high latency. They must respond to pings with a 512-byte payload. We would also prefer these to have uncorrelated failures, because if all of the outstanding pings fail for a block, then the data is permanently lost. So we want them to be geographically diverse, for example. I also prefer to use major commercial sites that can clearly bear the load, as opposed to e.g. someone’s cell phone (who may even have metered bandwidth). These are actually hard to identify from the full data set, particularly the last criterion, but it is not hard to find candidates by hand. I produced the list of ∼75 hosts by searching for “most popular websites in Madagascar” (etc.) and manually pinging them to make sure the criteria are met, particularly the latency. Many sites worldwide use content networks (or are simply hosted in the United States) and so they are much faster than the speed of light would suggest. I found that database-backed sites (like e-commerce pages) were less likely to be on content networks than e.g. news sites, which makes sense.

Implementing this block device is tricky: We can only process a read or write at the moment a ping returns from the network.

Blocks. Each block contains a sequence and version counter, as well as the set of outstanding pings (send time

[4] Named for the classic stop-motion penguin of the same name. Also as in “i ping u 2 store data thx 4 ur help”

[5] Try: `mkfs.vfat -F 12 -v -a -n "PINGU"`

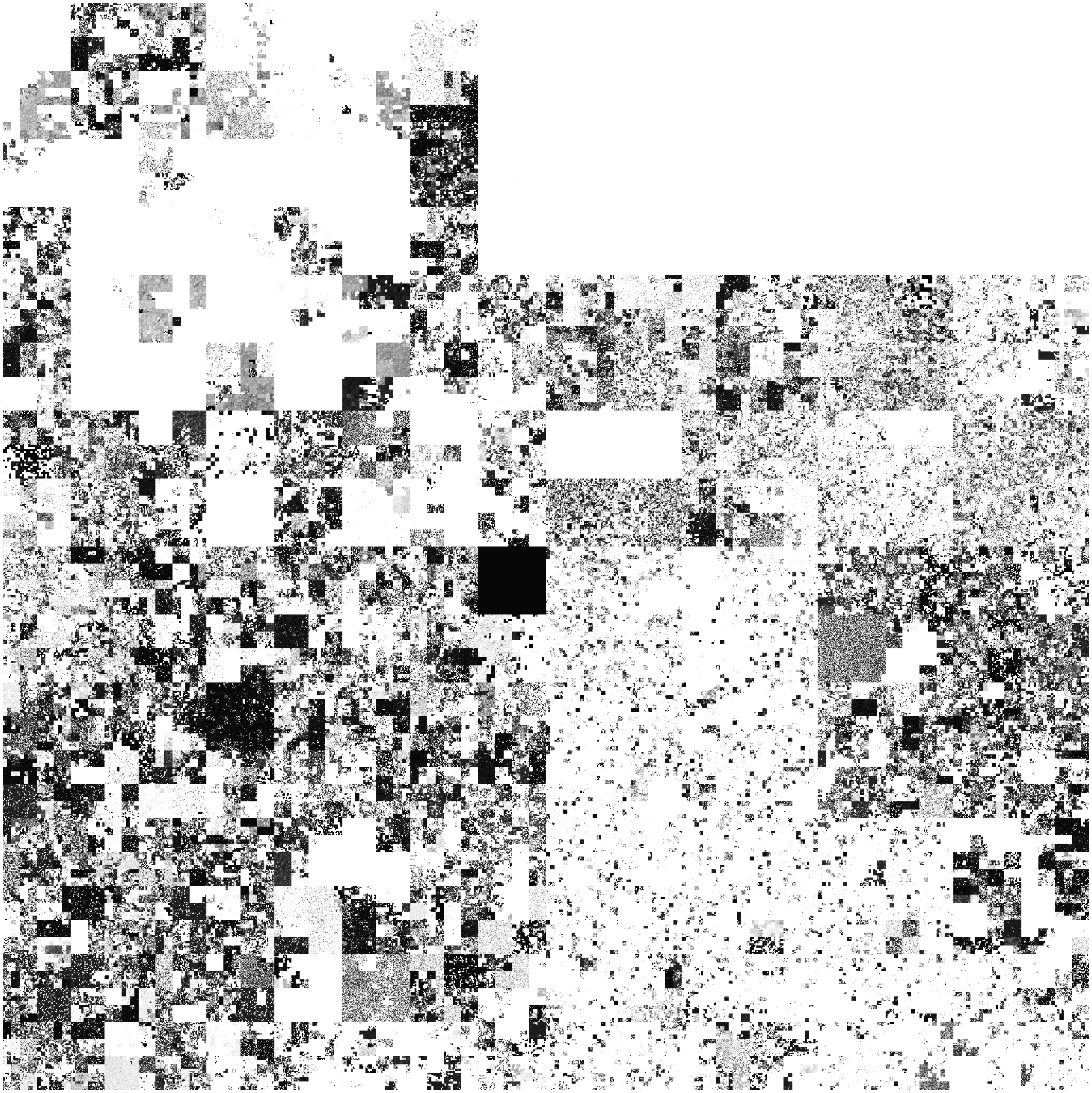
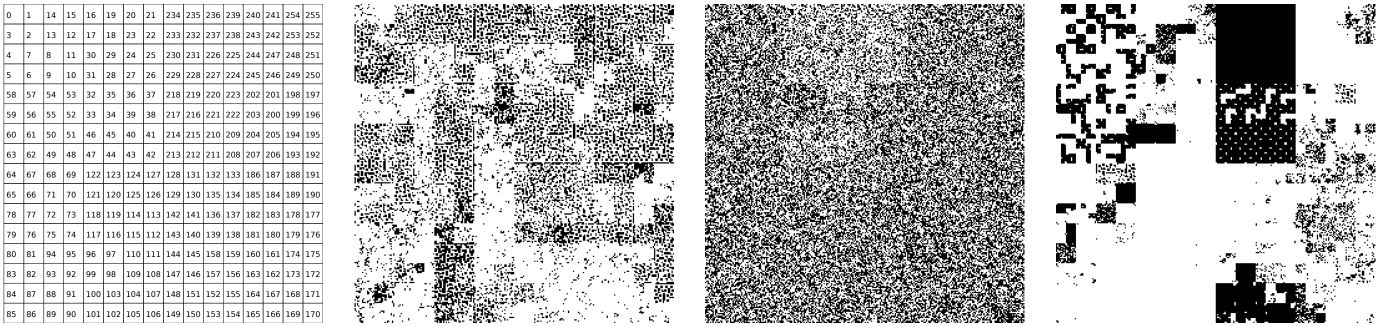


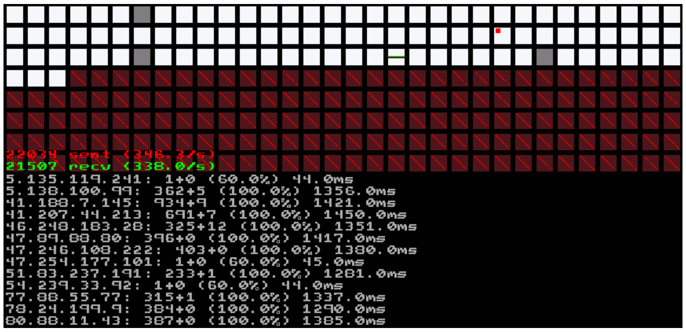
Figure 1: The results of pinging all 2^{32} IPv4 addresses in early 2022. The IP addresses are plotted along a 16-level Hilbert curve [9]. The full image is 65536 × 65536 pixels (and 4.2 Gigabits), which alas cannot be fit within the preposterously limited SIGBOVIK page and PDF size guidelines. This image is 2048 × 2048, so each pixel represents 32×32 hosts, with the level of grey giving the response rate (white = no response). There are several obvious patterns, which can be cross-referenced with Figure 2 to see the first octet of the IP address. Some interesting regions: There are almost no responses in the top-right region, which is `224.*` to `255.*`; these are the former "Class D" and "Class E" segments which are *multicast* and *reserved* respectively. There is a dark block at the center that almost always receives responses, which is from the `127.*` "loopback" addresses. This all makes sense. It is interesting to see how active regions allocate their space; some have a variety of distinctive patterns and others seem uniformly random (Figure 2). This way of plotting the address space is basically canonical, so it is a bit disappointing not to find any graphical messages. Like, how cool would it be to embed a micro QR code in some 16x16 subnet that says `IPv6sux`? It would be a little bit cool, is how cool.



0	1	14	15	16	19	20	21	234	235	236	239	240	241	254	255
3	2	13	12	17	18	23	22	233	232	237	238	243	242	253	252
4	7	8	11	30	29	24	25	230	231	226	225	244	247	248	251
5	6	9	10	31	28	27	26	229	228	227	224	245	246	249	250
58	57	54	53	32	35	36	37	218	219	220	223	202	201	198	197
59	56	55	52	33	34	39	38	217	216	221	222	203	200	199	196
60	61	50	51	46	45	40	41	214	215	210	209	204	205	194	195
63	62	49	48	47	44	43	42	213	212	211	208	207	206	193	192
64	67	68	69	122	123	124	127	128	131	132	133	186	187	188	191
65	66	71	70	121	120	125	126	129	130	135	134	185	184	189	190
78	77	72	73	118	119	114	113	142	141	136	137	182	183	178	177
79	76	75	74	117	116	115	112	143	140	139	138	181	180	179	176
80	81	94	95	96	97	110	111	144	145	158	159	160	161	174	175
83	82	93	92	99	98	109	108	147	146	157	156	163	162	173	172
84	87	88	91	100	103	104	107	148	151	152	155	164	167	168	171
85	86	89	90	101	102	105	106	149	150	153	154	165	166	169	170

(a) (b) (c) (d)

Figure 2: (a) An *internet legend*: The location of top-level octets in the Hilbert curve used to create Figure 1. Then, several 1 : 1 zoomed regions, showing how much the textures vary: (b) The subnet `213.6.*.*` (Palestine Telecommunications Company) shows some curious patterns of clumps or lines surrounded by whitespace, almost like a map of ancient city ruins. (c) The subnet `5.138.*.*` (Rostelecom Macroregional Branch South) is almost uniformly random, although it does look like it might be hiding a faint spooky skull at the top. (d) The subnet `45.195.*.*` (CloudInnovation) clearly has distinct subregions, which makes sense as it is an IP address management company. In that sense it Fractally resembles larger portions of the Internet. We also see some missing regions in the shape of Tetris pieces (Section 3).



```
22034 sent (346.3/s)
21507 recv (338.0/s)
5.135.119.241: 1+0 (60.0%) 44.0ms
5.138.100.99: 362+5 (100.0%) 1356.0ms
41.188.7.145: 934+9 (100.0%) 1421.0ms
41.207.44.213: 691+7 (100.0%) 1450.0ms
46.248.183.28: 325+12 (100.0%) 1351.0ms
47.89.88.80: 396+0 (100.0%) 1417.0ms
47.246.108.222: 403+0 (100.0%) 1380.0ms
47.254.177.101: 1+0 (60.0%) 45.0ms
51.83.237.191: 233+1 (100.0%) 1281.0ms
54.239.33.92: 1+0 (60.0%) 44.0ms
77.88.55.77: 315+1 (100.0%) 1337.0ms
78.24.199.9: 384+0 (100.0%) 1290.0ms
80.88.11.43: 387+0 (100.0%) 1385.0ms
```

Figure 3: Visualization of the `pingu` drive (truncated). The squares at the top are the data blocks; where white indicates a healthy block with a full complement of outstanding pings, and darker colors less so. The crossed-out blocks have not yet been written (and so store no data). The red dot indicates a block with an outstanding write, and the green bar the current block for the round-robin initialization. At the bottom, some of the hosts and their recent statistics.

and host IP, so that we can detect timeouts and update host stats). There is also a queue of pending reads and writes. The contents of the block is not stored.

Reading. A call to read a block inserts itself in a queue and then waits on a condition variable; it will not return until we receive a ping that belongs to that block.

Writing. A call to write is accompanied by some data (the caller has allocated it). These are also enqueued and wait synchronously until we receive a ping from the host and can process it. In the general case we cannot process a write without receiving the ping, because the write may only be to a portion of the block (and so we need to know the data outside that region). When we process the write we update the version counter so that we don't later use the data from any other outstanding redundant pings.

Hosts. With each host (IP address) we also keep track of its recent latency and reliability (exponentially-weighted moving averages) as well as a token bucket to prevent exceeding a prescribed number of pings per second to that host.

Processing. A single thread calls `select` to see if the socket is ready. For juggling we need to simultaneously be ready for both *reading* and *writing*. We then read a ping, and use its sequence number to route it to the correct block. The block validates the ping (if it has the wrong version it's just discarded, for example) and uses it to fulfill any outstanding reads (copying into their buffers and notifying the condition variable so those calls can return). We then process any writes to compute the updated data, and juggle the data back onto the network by sending pings until we are at the target redundancy (there will be at least one, since we just received one of the outstanding pings). For each outgoing ping we prefer a host with high latency, high reliability, and which has not recently been used. We also avoid using the same host more than once for a block, because if we lose all the outstanding pings, the data are forever lost.

Initialization. The loop just described is driven by the receipt of pings, so we also need to kick off the process by sending initial pings for each block. After each call to select we initialize a single block if it has not yet been, and has at least one outstanding write.

Visualization. The block device runs in userspace, but not in a way that supports a UI. To view the device while it's being used, I send text status updates to `nbdkit`'s debugging interface, and then pipe these to an SDL-based visualization (Figure 3). It shows the status of each block and statistics on each host, as well as read/write activity. It is fun to watch the process of formatting it for FAT12 and reading/writing files.

2.0.3 Results

Next we want to evaluate this Harder Drive according to various criteria. Our goal is not to create a drive that is "good" according to normal criteria like speed, but it is still interesting to benchmark it.

For each benchmark in this paper we will store a single file on the drive. The choice of file typically doesn't matter for the benchmark (which will compute "bytes per second" etc.) but it very much matters for the aesthetics of the project. In each case we'll choose a file that establishes a kind of "improper hierarchy" [16]. For `pingu`, we'll store RFC 792 [21], a 29,186-byte text file that describes ICMP, including the `ECHO` and `ECHO REPLY` messages with which we've constructed the drive.

Before benchmarking, we format the drive with a FAT12 filesystem and mount it (`-noatime`, etc.). We then `sync` and flush the kernel cache.[6] Flushing cache is very important, as these tiny drives easily fit entirely within the cache and appear to be very fast if you don't do this. Then, the benchmark writes the entire file to the test drive, syncs and clears cache again, and reads it back, comparing to make sure the correct bytes were written. We repeat this process over and over, for at least one minute (though some drives will only complete a single pass, taking much longer than a minute). The sync/flush between writing and reading is attributed to the write time, because this is when the writes are actually taking place.

Qualitative. This is a good Harder Drive. It solves a problem we don't have, which is to unreliably store a small amount of data in an even smaller amount of memory.[7] It treats latency as a desirable quantity, contrary to the usual preference. Implementing the drive was much more difficult than expected from back-of-the-envelope calculations.

Cost. The cost consists of an up-front cost (a computer and network interface; even a $35 Raspberry Pi should work fine) and an incremental cost per byte stored. This drive is unusual in that storage is derived from network bandwidth, which is measured in bytes per second. My home network is "1 Gigabit/sec" and $80/month ($\$3.04 \times 10^{-5}$ / second). Storing 51,200 bytes renders the connection otherwise unusable, so we assume this is close to the maximum storage. This is a cost of 0.156 cents per byte per month, which is 5.94×10^{-8} cents, or 59.4 nanodollars, per byte per second.

Longevity. Longevity is poor. The one-minute benchmark succeeds with 100% accuracy, but data will readily be lost if the drive is left running for several minutes. We can increase longevity by using hosts with higher latency, although this reduces speed.[8] Since the data are stored externally using untrusted hosts around the world, it is easy for adversaries to tamper with it by sending us back the wrong data. This could be mitigated with checksums or error correcting codes [23], although we want to avoid anything that might resemble "storing" the data locally (this is cheating). On the other hand, since we do not store data locally, this drive could be considered "non-volatile" in the sense that if we completely lose power and reboot, we can still recover the data as the pings are received from the network. Such a reboot would need to happen in less than about 100 milliseconds, though.

Speed. The drive is slow but tolerable. In the benchmark wrote and read the test file 15 times in one minute, and achieved 15,286 bytes/sec writing and 13,239 bytes/sec reading. Reads and writes are basically the same operation so this gives a small indication of the variance (high) as well. We can get better I/O performance by using hosts with lower latency, but this increases the (local) cost and decreases longevity.

Power. Power consumption is low. The up-front power cost for a computer and network connection is small (Raspberry Pi 3 is about 3 Watts). The data are actually stored externally, and if we were the only use of the Internet, a very significant amount of power would be consumed in transmission lines. In the benchmarked configuration, each 512-byte block has 4 outstanding pings, for which we assume a mean cable length of 1/4 Earth circumference (10 Mm). A copper connector like Cat6 UTP has nominal DC resistance of 84 Ω/km, so the loop resistance would be 840 MΩ, which is actually rather high. The math to figure out the power per byte eludes me (not to mention that undersea cables are usually fiber optics), but it is not trivial. A typical undersea data cable's excitation power is on the order of tens of kilowatts, with repeaters every 100 km or so. Fortunately, the total bandwidth of such cables is extremely high, with these 512-byte packets representing a minuscule fraction. Rather than try to multiply a big estimate by a small estimate, it is better to work from a known quantity: Assuming that the cost of the consumer internet connection also covers the marginal cost of the power in

[6] `echo 3 > /proc/sys/vm/drop_caches`

[7] I considered whether it would be possible to create a drive that used *no* memory for each additional block, and how I might even define/measure that. This lead me to a brief experiment with `compu`, which never graduated to a proper Harder Drive. This drive compiles the drive's contents as code, namely a large switch statement that is of the form `case ADDRESS: return DATA;` for each address in storage. The joke is of course that if you don't count the "code" towards memory, you can sneak memory into the code. Of course, to write to the drive, we need to rewrite the code and recompile it (this is done dynamically by forking `g++` and then loading the recompiled symbol with `dlopen` (which obviously uses memory)). I was hoping that the compiler would be able to do some clever optimizations on the switch statement, which might have led to some interesting developments. But the only ones I observed were the cases where the entire contents are the same byte, or where each address contains the low byte of the address as data. Otherwise it always just compiled as a table lookup, which is pretty uninspiring. For completeness, this drive was annoyingly fast in benchmarks (of course using its own source code as the benchmark test file): 6,119 bytes/sec writing; 10 Megabytes/sec reading.

[8] One easy approach is to deliberately incur additional overhead. For example by connecting to a VPN in Nigeria, I ensure a round-trip to Africa before the pings even make their way to the open Internet, which increases latency significantly. This reduces throughput to 2,948 bytes/sec., however.

these backbones, at $0.15c/kwH$, this seems to be at most 5.8 μWatts per byte-second.

Is rotational? One of our criteria will be whether the drive should set the `is_rotational` flag for `nbdkit` (Section 2.0.1). The inspiration for this drive (orbital chainsaws or radio towers around the world) would be rotational, but this drive is not. Although the initialization happens round-robin, due to the stochastic timing of the outstanding pings, the drive soon thereafter processes the blocks in a random order.

Harm to society. The drive is definitely harmful to my home network; whether that can be considered a positive or negative to society is left as an exercise for the reader. At the benchmark scale of 51,200 bytes, the effect on the broader internet is trivial, and I took care to not overwhelm any particular host. However, at scale this drive would be harmful to the shared infrastructure, and carries the moral hazard of "freeloading" off the willingness of hosts to reply to `ECHO` messages.

3 Tetris, the Soviet Mind Game

Sorry to remind you about Vladimir Putin's illegal invasion of Ukraine, but this section concerns a hard drive made from the best Russian (actually, Soviet) video game, Tetris [20].

Tetris is an inventory-management survival-horror game with 19 principal characters, each with its own story arc; they are:

Like all living things, these characters are made up of four individual pixels, or "blocks." By being confused about the fact that words can have multiple meanings, we can have an idea: Make a block device from these blocks, using their presence or absence in the playfield to store data. A Tetris board is 10 columns wide and 20 rows high. Even if we could use every one to store a bit, 200 bits is far too few to create a filesystem. Therefore we'll use an array (or if you will, a *Beowulf cluster*) of Tetris games to create the block device.

We will store a bit pattern in a Tetris game by playing a series of moves to create a specific pattern in the playfield. We can then read the data directly from that pattern. If we need to write a new pattern, we reset the game and begin again.

Each "line" of the playfield has 10 positions, each of which could have a block in it or not, so we could consider storing 10 bits. However, due to the rules of Tetris, if all of the cells are filled, then the line is cleared. This would make it impossible to store the pattern `1111111111`. It will also be impossible to store the pattern `0000000000`, because an empty line cannot support any pieces above it, so empty lines can only appear in some completely-empty prefix of the playfield. Additionally, observe that a Tetris board always has an even number of cells filled. We can only add 4 blocks by dropping a piece, or remove 10 blocks by clearing a line, which can only yield even numbers. So it will also benefit us to have one free cell per line for parity.

```
uint16_t NextRNG(uint16_t state) {
  uint16_t carry = ((state >> 9) ^ (state >> 1)) & 1;
  return (state >> 1) | (carry << 15);
}
```

Figure 4: The simplified code for NES Tetris's pseudorandom number generator, which resides at address `0xAB47` in the Tetris code. This is a 16-bit two-tap LFSR: The `carry` is the exclusive-or of the second and tenth least significant bits. We right-shift off the least significant bit and use the carry as the new most-significant bit.

A good choice is to use 8 cells per line for data, encoding a single byte, which is a nice round number.

We can't use the full height of the playfield, since we need some room in which to maneuver pieces. 8 is a convenient choice here as well (although more is possible). Each Tetris game will thus store 64 bits: Eight lines, each with eight bits. We'll use the venerable NES Tetris (Nintendo, 1989), which is also 8 bits.

3.1 Playing Tetris

Now the problem is: Given a blank board, what sequence of moves do we make in order to produce the target pattern?

This is not easy. To begin with, Tetris normally gives the player pieces at random. As anyone who plays Tetris knows, it can be very disruptive to your strategy when you don't get the piece you need for some time. The first step will be to reverse engineer the pseudorandom piece drop logic so that we can influence the sequence of pieces that are dropped.

3.1.1 Random pieces

The core of the piece drop logic is a 16-bit linear feedback shift register [8].[9] Equivalent C code is given in Figure 4. This 16-bit state is updated on every frame (and sometimes more; see below), and has period 32767.[10]

The pseudorandom state is extended with two additional bytes: One giving the last dropped piece (a piece is "dropped" into a queue so this is actually the "next piece" to the player) and the count of pieces dropped (mod 256). When the player places a piece, the routine at address `0x9907` uses the LFSR state and these two bytes to drop a new piece (and update the state):

```
RNGState NextPiece(RNGState s) {
  constexpr std::array<uint8_t , 8> PIECES = {
    0x02, 0x07, 0x08, 0x0A, 0x0B, 0x0E, 0x12,
    /* not used */ 0x02,
```

[9] Fittingly, Golomb was a pioneer in both shift registers *and* polyominoes, the latter which influenced Tetris itself!

[10] It is possible to create a 16-bit LFSR with a period of 65535, but this one is simply deficient. This is one of several small problems with the code. I hope to one day release a "hot fix" ROM that fixes this and other bugs and inefficiencies.

```
  };

  s.drop_count++;

  uint8_t a = (s.lfsr_hi + s.drop_count) & 7;

  if (a == 7 || PIECES[a] == s.last_drop) {
    // re-roll if out of bounds, or repeat
    s = NextRNG(s);
    // mod 7 forces in-bounds, but allows repeats
    a = ((s.lfsr_hi & 7) + s.last_drop) % 7;
  }

  s.last_drop = PIECES[a];
  return s;
}
```

It uses three bits of the RNG state to pick a random piece (there are 7 different shapes, and the game always drops a shape in the same orientation). If it rolls an 8, or if the selected piece is the same as the last one, then it re-rolls: Another update of the LFSR, and then a different weird procedure to pick the piece index. Here the result is `mod` 7, so it is always in bounds. The code only re-rolls once, so it is possible to drop the same piece twice in a row, just less unlikely.

Ideally we would be able to select a sequence of pieces that we want, and then force Tetris to give us those pieces. Since the LFSR update runs every frame, we can use a different number of frames while placing a piece, and get a different LFSR state at the point `NextPiece` is called. The LFSR is "pretty good," so we can easily cause the first roll to be whatever value we want by just waiting. In the worst possible case we need to pause 98 additional frames before seeing all 8 rolls (Figure 5), which is 1.6 seconds at the NES frame rate.

However, this does not work for re-rolls, which is the only way to get the same piece twice in a row. Even though we use the same "pretty good" LFSR to get the second pseudorandom number, two successive calls are highly correlated. There are only two possible new values for the high byte of the LFSR (`s.lfsr_hi`): `(s.lfsr_hi >> 1)` and `128 + (s.lfsr_hi >> 1)`. Worse, since these are congruent modulo 8, we really just have `(s.lfsr_hi >> 1)`.

As a result, even if you have complete control over the LFSR state (but not the previous piece nor drop count), there are a limited number of outcomes possible from the reroll. We can just inspect all the possible combinations of previous piece and drop count to see that with some there are at most 4 possible rerolls, and as few as 2. For example, if the previous piece is ▞ and 253 pieces have been dropped so far, then only ┻, ▞, and ┏ can result. So here it is possible to get two ▞ pieces in a row. But if the last piece was ▬, and 3 pieces have been dropped so far, then only ▀▄ and ┗ are possible from a re-roll. Since ▬ cannot result from the first roll and is not possible for the re-roll in this state, it is impossible to get two ▬ pieces in a row on the 3rd and 4th drop. All pieces other than ■ periodically have this problem.[11] Even when it is possible, it may require a long drought to get the LFSR in a rare working state.

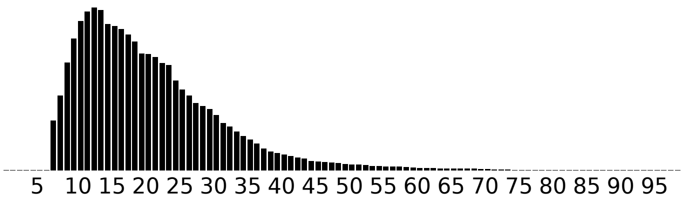


Figure 5: The maximum possible "drought" for the NES Tetris LFSR. This is the number of iterations before we see all possible rolls (all eight possible values for the low three bits), histogrammed over all meaningful start states (LFSR state and drop count). The minimal value is 8 (obviously, by pigeonhole) and rarely it can be as high as 99, but most of the time we have seen all rolls by a few dozen iterations.

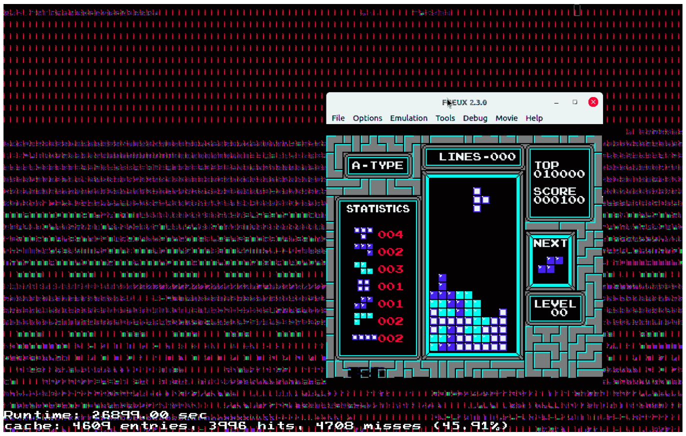
By manipulating the RNG we will be able to achieve any sequence of pieces, except if that sequence contains repeats. There is a similar consideration for the first two pieces of the game, which are generated on the same frame (current and next piece) and so only certain combinations are possible. To simplify this issue away, we begin by always playing the same starting sequence: ■0 ▟3 ▙6 ▛2 ▜7. This means to drop a ■ in column 0 (the leftmost column), ▟ in column 3 (the left edge of the piece goes in column 3), etc. This sequence clears two lines, leaving the board empty, with the only constraint now being that we cannot start with a ▜ piece. In fact we will always start with a ▚ piece in the leftmost column to make use of the modular plans described in the next section.

3.1.2 Planning

With a smart algorithm, it would be possible to plan moves within these constraints to generate moves for any byte pattern that we want to create. However, this is not an easy feat. Creating very sparse patterns (few `1` bits) is pretty challenging because pieces can only be placed on top of existing blocks (Figure 6).

Instead, the approach I took was to build a modular solution for each byte. We always place the board in a standard configuration (Figure 7) where a ▚ piece is in column 0. A portion of the board below it contains the encoded bytes (one per row) and parity (first two columns). We do not depend on the contents of this portion at all, so the first moves must hang off of the ▚ piece. For each byte, we need to come up with a series of moves that takes a starting configuration like this, encodes the byte (and its parity) in the bottom row, and then recreates the starting configuration with the ▚ piece moved up one line. These plans are also

[11] This is another example of a deficiency in the code that could easily be fixed. If we simply remove the instruction at `0x9925`, `AND #$07` (so that the re-roll just uses `(s.lfsr_hi + s.last_drop) % 7`) then *all* configurations can now produce all pieces upon re-roll. The modulus is computed with a loop, so this is not a strict efficiency improvement, but efficient alternatives with this property exist, like `((s.rng2 & 15) ^ s.last_drop) % 7`.



```
Runtime: 26899.00 sec
cache: 4609 entries, 3996 hits, 4708 misses (45.91%)
```

Figure 10: Playing a game of Tetris in the FCEUX emulator, in which we loaded the `tetris.nes` ROM from the `tetru` drive. The backdrop is a truncated portion of the visualization showing the contents of the 8,640 Tetris boards being emulated. The upper portion is the FAT-12 header and directory entries (it is mostly `0x00`) and the lower portion is the ROM data. I only have 100 points right now but as you can see I am gearing up to complete some sweet, high-scoring Tetrises.

for each block is 64 bits, or 8 bytes itself, suggesting a form of "content-addressed storage."

3.2.1 Results

In order to benchmark we need some file to write to the filesystem. A beautiful choice is the `tetris.nes` ROM file, which is 49,168 bytes. Though the minimal filesystem for FAT12 requires a device with 51,200 bytes, there is significant overhead from the filesystem header, directory entry, and so on. So to store this ROM we create a device with 69,120 bytes, which is 8,640 NES emulators. It is straightforward to scale to thousands of emulators, with the biggest challenge being to fit them all on the screen for some kind of visualization.

We benchmark as before, but then of course it is important for aesthetic reasons to load the ROM that we stored inside the drive to play a game of Tetris. Figure 10 shows this in practice.

Qualitative. This is a good Harder Drive. It solves a problem we don't have, which is that typical hard drive "blocks" are not made of actual "blocks," but Tetris players will recognize that the data on the drive is indeed made from blocks. It is very satisfying to watch the thousands of Tetris games drop pieces to encode the data.

Cost. Simulated on a computer, the up-front cost of storing data is fairly low. A basic 16-core desktop computer is about $1,800 in 2022. The software NES emulator uses 1,652,372 bytes of RAM for Tetris on a 64-bit machine, which is 650 instances per gigabyte. So we can store about 5200 bytes in about $4.37 worth of RAM,[16] which is 0.084 cents per byte. This could easily be improved; the emulators could be stored much more efficiently, because they are all emulating the same ROM. If we built this with actual Nintendo hardware, we would need one NES Console and one Tetris cartridge (or bootleg) per eight bytes. A used NES runs about $150 and a Tetris cartridge about $10. This is $20 per byte, which 24 thousand times more expensive.

Longevity. The stored data lasts indefinitely, as long as the computer (or Beowulf cluster of NES consoles) remains powered.

Speed. Writing the 49,168-byte test file `tetris.nes` takes 3 hours, 18 minutes and 52 seconds, for a data rate of 2.57 bytes/sec. Due to its caching nature, writing to the hard drive gets faster as it stores more data. Reading is much faster at 61,430 bytes/sec.

Power. On a modern computer, a gigabyte of RAM uses about 375 mW of power [10], so the marginal cost is 72 μW per byte. The NES console uses about 10 Watts, which would give us about 1.25 W per byte.

Is rotational? This drive `is_rotational`, because the Tetris pieces are rotated to place them in the correct orientation.

Harm to society. There is no harm to society for the software emulation. If built on real NES consoles, hoarding thousands of these machines and cartridges would be considered antisocial, as they are historic items that are in limited supply, and many people still enjoy collecting and using them for their intended purpose.

4 Cue the coronavirus

Speaking of using things for their intended purpose: Sorry to remind you about the worldwide pandemic still killing thousands of people every day, but this section concerns a hard drive made from COVID-19 tests.

SARS-CoV-2 is an RNA coronavirus first isolated in January 2020 [29]. Since it is highly contagious and can cause severe illness (especially in the immune-naïve), testing is an important part of the worldwide response. There are two widely available approaches to testing: Lateral flow antigen tests and PCR. Lateral flow tests detect a target molecule (e.g. the SARS-CoV-2 spike protein) by binding a tagged complementary molecule (antibody) to it as the sample flows along a capillary bed. This is awesome. The tests are fast and cheap. Polymerase Chain Reaction (PCR) [24] tests work by amplifying a target sequence of DNA exponentially. It heats and cools the sample in the presence of a heat-stable DNA polymerase (typically *Taq* polymerase, which was isolated from the thermophilic *Thermus aquaticus* bacterium) and a bubble-bath of nucleotides that can be used to create more DNA. Each thermal cycle first unzips a double-stranded molecule into two pieces, and then

[16] In 2022, a Corsair 16GB DDR4 DIMM is only $70.

reassembles each one, doubling the target. Properly, the tests are reverse transcription PCR [4], because first we need to turn the viral RNA into DNA. PCR tests are more sensitive (they can detect a *single* molecule) and specific (they detect a particular genetic sequence). This is also awesome.

Cue is a bougie COVID-19 test that launched in 2021. The test consists of a reusable reader ($200) and a single-use cartridge ($65 *each*!). Notwithstanding the eye-popping expense, the system is pretty good. My employer provides these tests for free (!), so I started collecting the used cartridges over the course of several months, and soliciting them from my friends as well.

The cartridge itself (Figure 12) is fairly ingenious and deserves to be disassembled.[17] When you stick the nose swab into the cartridge, it of course is delivering the snotty sample, but the insertion force also mechanically actuates a number of plastic thorns which pierce foil seals on reagent ampules, allowing them to start their thing. The assay is described as "nucleotide amplification," which would suggest something like RT-PCR, but since the cartridge does not significantly change temperature during a test, it is probably not literally PCR. LAMP [19] is a similar technique which is *isothermal* and seems like a credible choice. Again it would be preceded by a reverse transcription step to turn RNA into DNA. These things are all awesome and deserve to be learned about; for example did you know that the most frequently used reverse transcriptases (which turns RNA into DNA to begin amplification) were isolated from RNA viruses (which those sneaky bastards use to turn their own RNA into DNA so that it can be transcribed by the host)? So now we're using virus machinery to detect and fight other viruses? Hell yeah we are!

If you pull off all the chemistry pieces from the cartridge's endoskeleton, you'll also find a tiny 8-pin microchip onboard. And if you have a microscope you can read that it says ST 24C04WP, which is a serial EEPROM [25]. An EEPROM is a programmable ROM (so it is "read-only" but also writable?). It is probably used to store the cartridge's serial number, expiration, what kind of test it is, and maybe calibration data. This stuff would be pretty small, so it's no surprise that the chip can only store 512 bytes. A dump of one of these ROMs is in Figure 11.

The EEPROM is an I^2C device, so I could use preexisting code to send commands to it. Reading the EEPROM is normal difficulty. I found writing to be more like "hurt me plenty" difficulty: When you write a line, the EEPROM drops off the bus temporarily (it may need to do this because it is internally stepping up voltage for the erase operation). You then have to either wait "enough time" or poll it to see when it's ready to write the next line. What I did is to repeatedly try to read back the same line we just wrote (this also allows us to check that the data were successfully written). However, sometimes the chip would come online *during* the read command, producing unpredictable results. Basically you have to be tolerant of errors in some situations, but not *too* tolerant, or else you don't detect real failures. It gives me some sympathy for terrible dedicated EEPROM programmers I have used [15].

```
00 00 b6 98 d0 19 5a 45 3b ef a1 d1 41 37 e6 ec | ......ZE;...A7..
49 91 6f 3b ab 7d 59 32 8f d4 e1 a6 33 bf 66 4b | I.o;.}Y2....3.fK
d6 fb 6a 9d f6 d4 89 72 a4 3d 8a 5b 62 10 4b 07 | ..j....r.=.[b.K.
d4 c3 15 52 01 d9 20 c1 97 87 4d c2 34 df 2a af | ...R......M.4.*.
cc 05 01 00 00 00 2e 00 0a 19 08 c6 ad d6 40 10 | ..............@.
13 20 a3 82 01 30 80 d6 89 99 06 3a 06 32 30 39 | .....0.....:.209
34 35 48 12 11 9a 01 0e 08 02 15 00 80 88 c5 20 | 45H.............
01 2d 00 00 c8 c3 ff ff ff ff ff ff ff ff ff ff | .-..............
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
```

Figure 11: ROM dump from a Cue COVID-19 test's onboard 512-byte EEPROM. After soldering tiny wires onto the tiny pins and writing a driver for it, I had hoped to see a secret message congratulating me on my steady hands and the beginning of an Alternate Reality Game whose prize was the inheritance of an eccentric billionaire (but who's got time for that?). Alas there is nothing that can be easily deciphered on here other than perhaps `20945H`. Note how much of the EEPROM is unused, but I'm glad they sprung for 512 bytes, or else this project would not have been so *feasible*.

4.1 Harder Drive: Cu

Having committed to the naming scheme where I replace some of the last letters of the thing with the letter u, it seems the best name for this drive is `Cu`. For one thing, this is the chemical symbol for copper, and the drive uses copper to function.

With the ability to read and write a single Cue cartridge, the remainder is just a matter of straightforward engineering and tedious manual labor. Of course, you want to do all of this on a manufactured printed circuit board (Figure 13). The job here is to make it possible to individually address a single EEPROM to read and write its data. Though I^2C does support multiple devices on the same bus, these chips all have the same I^2C address and so they would all try to reply to the same commands. The ST 24C04WP EEPROM does have "chip enable" pins that would allow it to share a bus with others, by selectively enabling only the chip of interest. Unfortunately, these pins are not connected to any of the exposed connectors on the cartridge. Instead, I use a bus switch (which is basically this same "chip enable"

[17] I recommend only disassembling a "negative" test, in case that is not obvious. The typical reagents used in RT-LAMP are not particularly dangerous; for example *Bst* polymerase is "not hazardous" according to OSHA, although it may be an "eye irritant" [2].

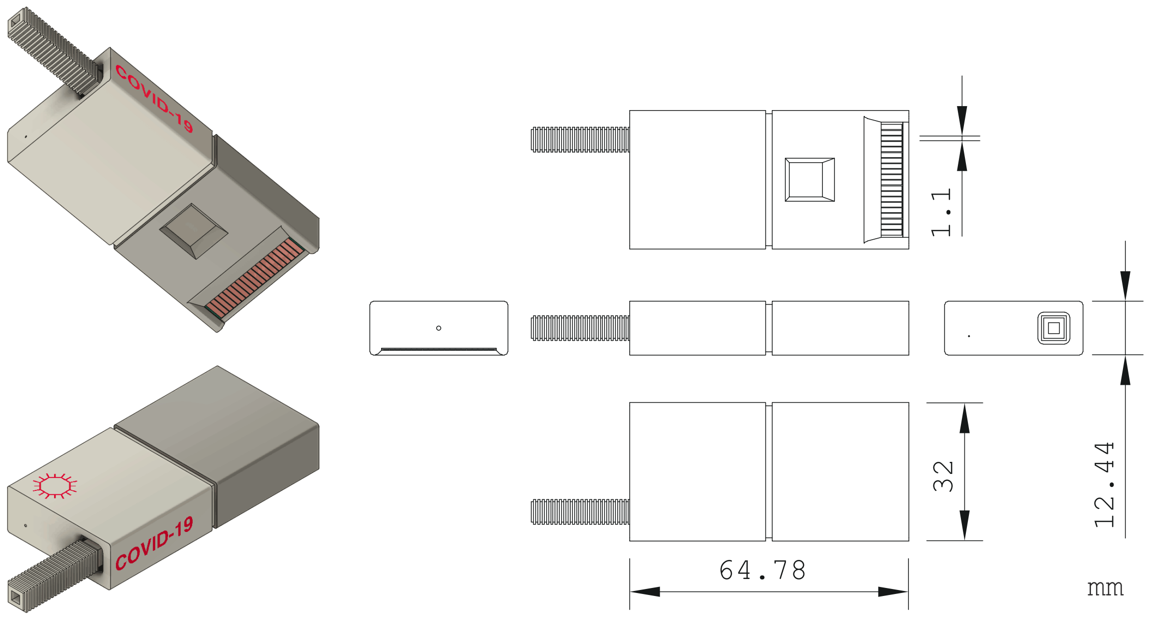


Figure 12: Mechanical drawing of the Cue COVID-19 test cartridge. The protruding stick is the nasal swab, which is permanently captured during use with zip-tie–like ratcheting. The card edge connector is the low tolerance piece here, whose small size (0.05 inch pitch with 1.1mm fingers) requires special consideration for mounting and soldering. (In its normal usage, this connector mates with some spring-loaded pins when the cartridge is inserted in the Cue reader.)

circuitry) to connect each EEPROM to the I^{2}C bus. Each SN74CBTLV3125 is a quad bus switch, so I can switch the two I^{2}C lines (SDA, SCL) for two Cue cartridges with each chip. Then, we can select one of 8 cartridges (a single daughter board) with a demultiplexor, which takes 3 address bits and sets exactly one of its 8 output lines to `0`. For decorative purposes, I associate a colored LED with each cartridge; this LED foolishly ends up accounting for most of the components on the board and most of the assembly time, since I also have to build logical NOT gates (demultiplexor outputs logical `0`). Finally, the cartridges themselves are very tricky to incorporate. They have very small plated connectors that normally mate with spring-loaded pins in the reader, but that component is not readily available (and would probably be expensive). Instead, I mount it at 90° through a hole in the PCB, where the PCB has its own matching edge connector made with castellated holes. I also 3D printed a plastic jig that could hold the cartridge at the right angle during soldering. With generous acid flux and a steady hand, soldering these worked quite well. Only 4 pins need to be connected (3v3, GND, SDA, SCL) but I also soldered some distal pins, since these joints are the only mechanical connections for the cartridges.

The motherboard has its own demultiplexor to select the daughter board, as well as an ad hoc pair of "group selector" pins wired directly to GPIO. Together it supports 7-bit addresses, for up to 128 cartridges, which is 64 kilobytes. I did not collect enough used tests to fill the address space, but I did connect 72 of them, which is enough to do something interesting at least! Except . . .

4.2 Failure!

I blew it! Literally! On the evening of the SIGBOVIK deadline, in an attempt to be simultaneously *expedient* and *careful*, I soldered the Cu motherboard while it was plugged in, and fully toasted it and the connected Raspberry Pi. My best guess is that the soldering iron had a very different idea of "ground" than the device under test. It made an upsetting pop noise, an upsetting burn smell, an upsetting spark and smoke sight, an upsettingly warm touch,[18] and it made it impractical to fix before the paper is due. You can at least see what the tabletop device looks like in Figure 14. It should be possible to replace the Pi and motherboard, so perhaps the by the accompanying video or talk I will be able to finish the task and get some benchmark numbers.

[18] I did not attempt to taste it; the board is not RoHS-compliant due to the copious lead used. Despite the panoply of upsetting sensations, the obviousness of the failure was a blessing that saved me time trying to debug! Even if I plug the Pi in on its own, it makes a pathetic, obviously unhealthy whine. It is so dead.

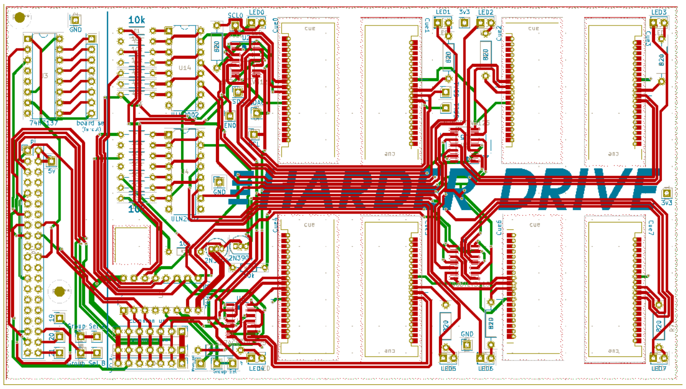


Figure 13: The two-layer printed circuit board for the Cue drive. Because the boards must be ordered in quantity, one board contains the layout for both the motherboard (used once) and daughter boards (used many times). On the far left is the motherboard, for example the header for interfacing with the Raspberry Pi and the 3:8 demultiplexor for selecting a daughter board. These parts are only populated once. The remainder is the daughter board: Eight cutouts for cue cartridges mounted at 90° with castellated edges; an LED and support logic for each; the surface-mount bus switch ICs; another 3:8 demultiplexor for selecting the cartridge on this board. The design can accommodate 16 daughter boards, each with 8 cartridges, for a total of 64 kilobytes.

4.2.1 Results

We need a file to store on the drive to tie the knot and to run the benchmark. A beautiful choice here is the genetic sequence of SARS-CoV-2 (ancestral) [30] from GenBank [5]. This is a 77343 base-pair sequence; it would be too large to store in ASCII.[19] Each nucleotide is only two bits of information, though, so we pack four of these into each byte for only 19336 bytes.

Qualitative. This is a decent Harder Drive. It solves a problem we don't have, which is what to do with all those COVID-19 tests that we'd otherwise just throw away? It took significant effort to create, although most of the difficulty was from problems (e.g. how to solder these tiny pins) that are not interesting from a computer science perspective. EEPROMs are fundamentally data-storage devices, so this usage of them cannot be considered clever, but arraying dozens of them to create a modest-sized non-volatile memory that could be easily replaced with a single 30-cent NAND Flash IC is at least "very silly." Mucho demerits because I broke it during the final assembly.

Cost. The cost is significant. The up-front cost is a Raspberry Pi 3 (nominally $35), accessories, and a demultiplexor IC ($0.60), plus scrap plywood for mounting. Then, per board, we have the following bill of materials:

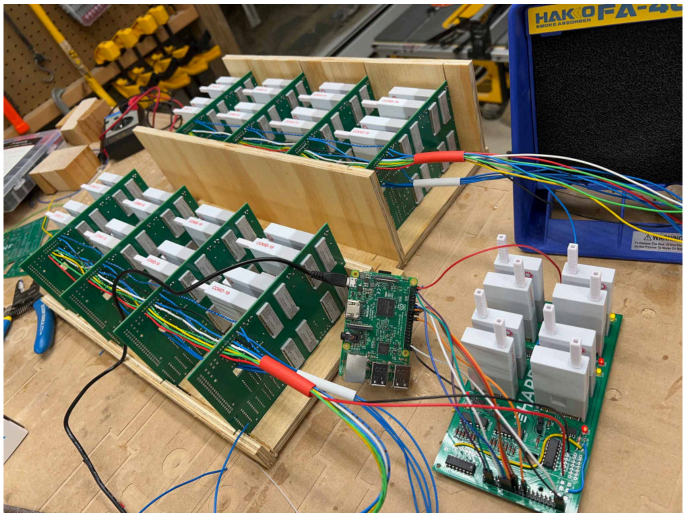


Figure 14: Assembled `Cu` drive with 72 Cue cartridges. Imagine trying to explain to someone that this homemade thing that has "COVID-19" written on it 72 times, and has got all sorts of colored wires everywhere, is not some instrument of bioterror. In fact it does not even drive hard: In my rush to meet the preposterously strict SIGBOVIK deadline, I fried the Raspberry Pi *and* motherboard (perhaps you can see that multiple LEDs are lit on the motherboard, which clearly violates invariants). Fortunately due to its modular design, it can likely be fixed with a few more hours of manual labor.

Qty.	Part no.	Description	Price ea.	Total
8	Cue L2900006	Used COVID-19 Test	$65.00	$520.00
1	custom	2-layer PCB 162x92mm	$5.766	$5.766
2	497-2340-5-ND	Transistor array IC	$0.5768	$1.154
4	SN74CBTLV3125	Bus switch IC	$0.6636	$2.654
2	2N3904	NPN BJT transistor	$0.09	$0.18
20	jellybean	10kΩ resistor	$0.0155	$0.31
8	jellybean	845Ω resistor	$0.02428	$0.194
8	jellybean	3mm LED 2v 20mA	$0.01499	$0.12
1	CD74HC137E	3:8 demultiplexor IC	$0.6048	$0.605
				$530.98

This does not include consumables like solder and hookup wire, nor the considerable time to assemble each board (about 1 hour with practice).

We need 13 boards to store a full FAT12 filesystem, for a total cost of $6,936.04. The marginal price per byte is 12.96 cents.

Longevity. This is the only drive considered where data are retained when powered down. The M24C04-W EEPROM is rated for 200 years of data retention, and 4 million write cycles [25]. At the current pace, this is likely to outlast the human race.

Speed. Unknown as of publication! As described in Section 4.2, the motherboard was damaged on the eve of the deadline and no benchmark was conducted. Reading the EEPROM is fast, but writing a block takes a few hundred milliseconds. The speed is expected to be high (for the drives considered here).

Power. The up-front power cost is low; we need to power the Raspberry Pi and various chips on the boards. Only one of the decorative LEDs is lit at a time, using about

[19] Plus, I'm a biology noob and I may just be missing something, but GenBank uses "T" in this sequence even though it should be U (uracil) in RNA, which seems very non-canonical?

1 mW. The total is about 3 Watts. The marginal power cost is excellent: On the Cue cartridge, only the EEPROM is powered. During standby it uses no more than 3 μA at 3.3 V, which is 9.9 μW for 512 bytes, or 19.3 nW per byte.

Is rotational? This drive is not rotational; it provides us SSD-like random access to the Cue cartridges, and the EEPROMs on board allow random access to each line of data.

Harm to society. Arguably, the drive is beneficial to society. First, it is built mostly from trash. Second, coronavirus testing prevents death or other hardship by informing infected people that they may be contagious; at a minimum it is good for the spirit by facilitating lower-anxiety gatherings. Finally, since the tests contain captured body fluids, this adds an all-too-rare "human element" to computing.

5 Other hard drives we really didn't need

Here are some things I hate: (1) The name of TDAmeritrade's stock trading app, which is "thinkorswim." This is of course a play on the idiom to "sink or swim," meaning metaphorically to toss someone into deep water to survive by their own efforts, or else drown. The analogy is certainly apt for an app that lets consumers trade derivatives, but the obvious problem here is that if it is "think or swim," then we are now asking the subject to survive by their own efforts (swim) or else "think"? Huh? Or is it that they must think carefully about their trades, or else they will survive? Wha? (2) Poison ivy. This plant has no purpose other than to make you itch. It doesn't even get anything out of that trick, since I wasn't going to eat it anyway. Nevertheless it spreads. (3) Cryptocurrency. I have no objection to the use of cryptography in finance, but there aren't enough vomiting emojis in Unicode to appropriately react to the current hype. Cryptocurrency significantly harms the planet while taking advantage of many people's technical and financial illiteracy.[20]

Nonetheless, the common prefix between "block device" and "blockchain" is hard to avoid noticing, and a head-to-head comparison may be instructive. So I put on incognito mode, a VPN, an N-95 mask and six condoms in order to research some numbers for this section.

Bitcoin is "append-only" by design, so it does not have the same abstraction as other Harder Drives. For comparison sake, we consider a usage where the head of the blockchain contains the full data; a write is accomplished by mining a new block and a read is accomplished by reading from the current head. For Bitcoin, the block size is 1 Mb, and the network automatically adjusts to mine a single block every ten minutes. I did *not* actually implement this drive, both because of the gag reflex and because I do not have that kind of money!

Qualitative. Despite hating it, I must admit that Bitcoin meets the criteria for a Harder Drive pretty well. It solves a problem that we don't have, by imagining a world where we cannot agree on a small set of trustworthy parties, a majority of which must be acting in good faith. Its approach is elegant in the small but for its obvious fatal flaws, and comically absurd if taken to its logical extreme. It is impressively inefficient, and grows less efficient over time. In short, it would make a solid SIGBOVIK paper. The only problem is that people are actually using it in seriousness, and the social problems that result from the value it has attained.

Cost. The cost is extremely high. The reward for mining one Bitcoin is currently 6.25 BTC, plus an average of about 0.97 BTC in transaction fees, which totals $342,000 in March 2022. This gives us an approximate upper bound on the cost to mine (by assuming the marginal cost is profitable) a block, which is 34.2 cents per byte. This does not include the up-front cost of hardware and facilities, which is of course monumental.[21]

Longevity. The data has excellent longevity, in fact, it is impossible to erase previous data once written. Of course, "forks" of the chain can make it unclear what version of the data is correct, or if $>$ 50% of the untrusted miners

[20] Note to cryptocurrency apologists: This short section does not have room for a full criticism, nor would such a thing be "fun" enough for SIGBOVIK. Briefly, there are five principal problems. (1) Proof of Work is incredibly wasteful (see the stats below; this is just *one* of the cryptocurrencies). Of course I am aware of "Proof of Stake." I remain very skeptical that miners with large capital investments in (otherwise useless) ASICs will be willing to salvage them, but I would gladly celebrate this and by all means, please do make this happen. (2) I believe that regulation of finance is *good*, both formal regulation with law and self-regulatory organizations like FINRA, as well as informal practices like rolling back erroneous transactions or returning stolen funds, which are regulated indirectly by the desire to maintain valuable public reputations. Unregulated markets have many problems (insider trading, etc.) and avoiding regulation mostly seems to be useful for tax evasion and other crimes. (3) In attempt to avoid "decentralization", control is nonetheless effectively centralized in the hands of a small number of actors anyway (large-capacity miners and exchanges). However, these actors are set up as adversarial, or at best as some kind of wild-West "disruptors." I'd trust these skeezy guys way less than I trust banks, and rightly so: They routinely front-run transactions, just as one example. (4) The space is riddled with Ponzi schemes and scams, as exemplified (but certainly not limited to) NFTs. This is plainly immoral. (5) Cryptocurrency aficionados are insufferable, presumably because they feel like they need to convince you to get in on their tokens so that they grow in value (which presumably they hope to then sell to get real money). I get an enormous amount of cryptocurrency spam. The only words I've muted on Twitter other than five green Unicode squares are cryptocurrency terms, and this has improved the experience greatly. If you are a cryptocurrency aficionado reading this that feels like you need to "educate" me about how I am misinformed (despite what I can clearly see in front of me!), case in point. That said, I will happily discuss interesting ideas with informed computer scientists over beer.

[21] Nor the surrounding apparatus like "bitcoin ATMs" and "crypto exchanges" (the kind of stuff that apologists are talking about when they tell you that "regular money uses a lot of electricity too!"), although it is probably not fair to count these as part of the cost when used as a pure data storage mechanism.

disagree, they can change the data at will.

Speed. The network is slow, although not the slowest considered here. It takes an expected ten minutes to write 1 Mb of data. This is a data rate of 1,747 bytes/sec, approximately the speed of a 14.4 kbaud modem.

Power. The power usage is incredibly high. Mining Bitcoins uses about 0.31% of the entire world's energy production, 15.74 GW [6]. Remember that this does not solve any interesting computational problems or accomplish anything useful; the only purpose is to create an expensive waste of power in order to avoid trusting a bank or set of banks. To store one megabyte on an ongoing basis, this is 15.74 kW per byte.

Is rotational? It's not even rotational!

Harm to society. The harm to society is significant. Aside from the catastrophic waste of resources, the primary use case is speculation (at best morally neutral, but probably tends to harm small investors). As a slow, expensive, non-atomic yet irrevocable payment mechanism, they are best suited for extortive transactions like Ransomware.

6 Conclusion

In this paper, we decided that sometimes it's more fun to do things the hard way, and then did so. Using several different techniques and some needless digressions, we created block devices that could support small filesystems, which then could host a fitting file. Each filesystem was bad when considered as a regular hard drive, but good when considered as a Harder Drive. We also compared these drives to the most popular cryptocurrency. The idea was to make the point that cryptocurrency is so egregiously bad that it resembles a "SIGBOVIK joke gone wrong" more than something one would make on purpose. This part may not have been as fun.

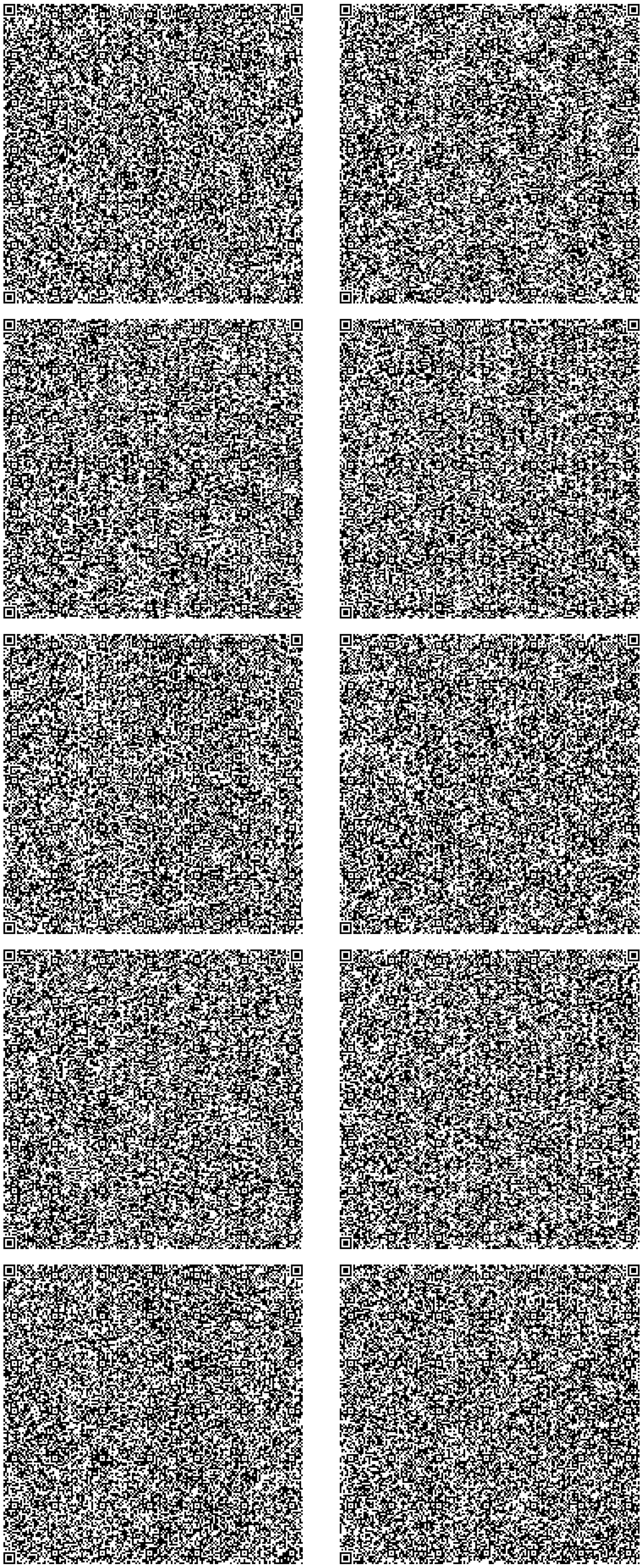
Acknowledgements.

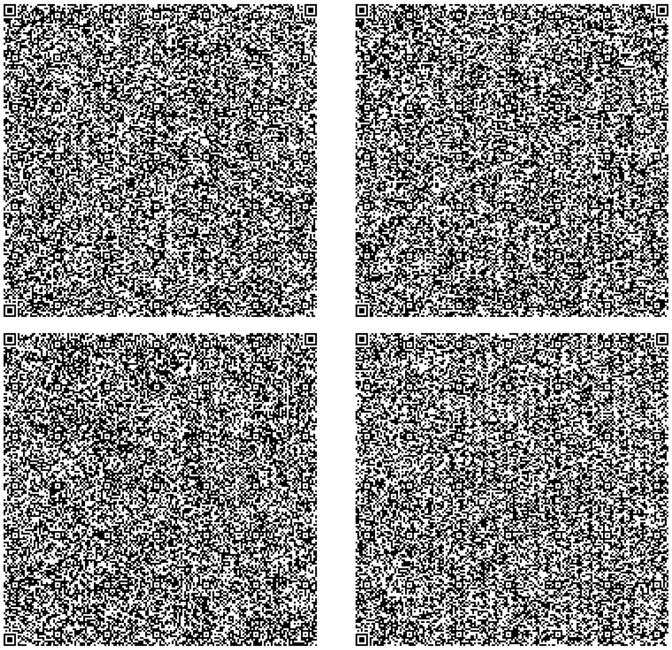
I used `nbdkit` [3] to create block devices, which was a much sounder idea than writing my own kernel drivers. I still found it very easy to render Linux unusable. The code for sending and receiving pings was adapted from `liboping` [7].

Thanks to Rose, William, Sophia, Reed, Jessica, Max, and Finn for donating their nasal swabs to the project. I especially appreciate that they gave me these samples without any information about what I was even doing with them. That's true friendship. By the way, human cloning is possible now. Just sayin'.

If you have a computer connected to the internet, then I attempted to send it a message during the course of this project. If your computer responded, then thank you for configuring it that way. Similarly, this paper would like to acknowledge any TCP SYN packets that are sent to it (but it cannot, for it is simply a paper).

Finally I would like to thank the SIGBOVIK program committee for letting me store this file in the proceedings:





References

[1] adelikat et al. FCEUX, the all in one NES/Famicom emulator. `fceux.com`.

[2] New England BioLabs. Material Safety Data Sheet: Bst 2.0 DNA polymerase, m0537. November 2019. `https://www.neb.com/products/m0537-bst-20-dna-polymerase`.

[3] Eric Blake, Richard W. M. Jones, et al. nbdkit, 2022. `gitlab.com/nbdkit`.

[4] Stephen A Bustin. Absolute quantification of mRNA using real-time reverse transcription polymerase chain reaction assays. *Journal of molecular endocrinology*, 25(2):169–193, 2000.

[5] K. Clark, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and E. W. Sayers. GenBank. *Nucleic Acids Research*, 44(D1):67–72, January 2016.

[6] Cambridge Centre for Alternative Finance. Cambridge Bitcoin electricity consumption index, March 2022. `ccaf.io/cbeci/index`.

[7] Florian Forster. octo's ping library, 2017. `noping.cc`.

[8] Solomon Wolf Golomb and Lloyd R. Welch. Shift register sequences, 1967.

[9] David Hilbert. Ueber die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen*, pages 459–460, 1891.

[10] Micron Technology Inc. How much power does memory use?, 2019. `www.crucial.com/support/articles-faq-memory/how-much-power-does-memory-use`.

[11] Jim McCann and Tom Murphy, VII. The fluint8 software integer library. In *A Record of the Proceedings of SIGBOVIK 2018*, pages 125–128, April 2018. `sigbovik.org/2018`.

[12] Tom Murphy, VII. The first level of Super Mario Bros. is easy with lexicographic orderings and time travel. After that it gets a little tricky. *SIGBOVIK*, pages 112–133, April 2013.

[13] Tom Murphy, VII. New results in $^k/_n$ Power-Hours. *SIGBOVIK*, pages 5–14, April 2014.

[14] Tom Murphy, VII. `zm~~ # printy# c with abc!`. *SIGBOVIK*, pages 129–148, April 2017.

[15] Tom Murphy, VII. Making of ”Reverse emulating the NES. . . ”, May 2018. `youtu.be/hTlNVUmBA28`.

[16] Tom Murphy, VII. Reverse emulating the NES to give it SUPER POWERS, May 2018. `youtu.be/ar9WRwCiSr0`.

[17] Tom Murphy, VII. Elo World: A framework for benchmarking weak chess algorithms. In *A Record of the Proceedings of SIGBOVIK 2019*. ACH, April 2019. `sigbovik.org/2019`.

[18] Tom Murphy, VII. NaN gates and flip FLOPS. In *A Record of the Proceedings of SIGBOVIK 2019*, April 2019. `sigbovik.org/2019`.

[19] Tsugunori Notomi, Hiroto Okayama, Harumi Masubuchi, Toshihiro Yonekawa, Keiko Watanabe, Nobuyuki Amino, and Tetsu Hase. Loop-mediated isothermal amplification of DNA. *Nucleic acids research*, 28(12):e63–e63, 2000.

[20] Alexey Pajitnov. Tetris, June 1984.

[21] Jon Postel. Internet control message protocol. STD 5, RFC Editor, September 1981. RFC 792, `www.rfc-editor.org/rfc/rfc792.txt`.

[22] Jon Postel. Internet protocol. STD 5, RFC Editor, September 1981. RFC 791, `www.rfc-editor.org/rfc/rfc791.txt`.

[23] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[24] Randall K Saiki, Stephen Scharf, Fred Faloona, Kary B Mullis, Glenn T Horn, Henry A Erlich, and Norman Arnheim. Enzymatic amplification of β-globin genomic sequences and restriction site analysis for diagnosis of sickle cell anemia. *Science*, 230(4732):1350–1354, 1985.

[25] STMicroelectronics. M24C04-W M24C04-R M24C04-F datasheet, October 2017. `www.st.com/resource/en/datasheet/m24c04-w.pdf`.

[26] Wikipedia. File allocation table, 2022. `en.wikipedia.org/wiki/File_Allocation_Table#FAT12`.

[27] Wikipedia. Ping of death, 2022. `en.wikipedia.org/wiki/Ping_of_death`.

[28] Wikipedia. Smurf attack, 2022. `en.wikipedia.org/wiki/Smurf_attack`.

[29] F. Wu, S. Zhao, B. Yu, Y. M. Chen, W. Wang, Z. G. Song, Y. Hu, Z. W. Tao, J. H. Tian, Y. Y. Pei, M. L. Yuan, Y. L. Zhang, F. H. Dai, Y. Liu, Q. M. Wang, J. J. Zheng, L. Xu, E. C. Holmes, and Y. Z. Zhang. A new coronavirus associated with human respiratory disease in China. *Nature*, 579(7798):265–269, March 2020.

[30] F. Wu, S. Zhao, B. Yu, Y. M. Chen, W. Wang, Z. G. Song, Y. Hu, Z. W. Tao, J. H. Tian, Y. Y. Pei, M. L. Yuan, Y. L. Zhang, F. H. Dai, Y. Liu, Q. M. Wang, J. J. Zheng, L. Xu, E. C. Holmes, and Y. Z. Zhang. Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1, complete genome, January 2020. GenBank MN908947.3, `www.ncbi.nlm.nih.gov/nuccore/MN908947`.

Just a Regular Paper

Rob Burr
Machiavellian Institute of Thievery
Ohio, USA

Abstract

This is just a regular paper. There is nothing of interest here, nor should anyone feel like something is out of the ordinary. Oh, the ski mask, you ask? That's just because it's a frigid 60 degrees outside, don't worry. Seriously, you're acting a bit too on edge from seeing a guy with a ski mask on. Don't mind the large bag I have either, I just like to use it to... carry things sometimes. Yeah.

1 Introduction

Hello, everyone, I am just another person here! This is what you do during an introduction, right? Why do you want to know more about me, huh? You tryna start something? Yeah, that's what I thought. Back down, bud.

2 Methods

What do you mean, methods? I'm not trying to start anythin'! I'm just another guy! Just another regular dude! Like, look at this graph:

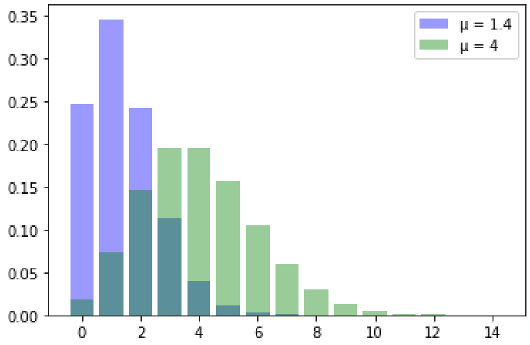


Figure 1. Just a normal graph!

Ain't nothin' wrong with this graph! It's a cool graph! I can use matplotlib!

3 Discussion

Okay, this is getting tiring. *pulls out weapon* EVERYBODY, LISTEN UP! THIS IS A ROBBERY! I WANT EVERYBODY TO GET ON THE GROUND, NOW! I SAID, EVERYBODY, GET ON THE GROUND, *NOW!!!!!* NOW LISTEN CLOSELY: I WANT EVERYBODY TO TAKE THEIR WALLETS AND TOSS THEM TOWARDS ME. YUP, HAND THEM OVER. DON'T TRY TO BE A HERO, OR THIS HUMAN-COMPUTER INTERACTION PAPER OVER HERE GETS BURNED. (A person shouts, "Boooo, pick a better hostage!") ALRIGHT, FINE, I WILL! THIS *MACHINE LEARNING* PAPER IS GOING TO GET BURNED! (everybody gasps) *turns to teller* OK, NOW I WANT $200,000 OF UNMARKED, NONCONSECUTIVE $20 BILLS NOW.

Time (s)	Money Collected ($)
0	0
10	0
20	8,000
30	32,000
40	64,000
50	81,000
60	124,000
70	156,000
80	200,000

Table 1. The money that has been placed in the bag, now.

OKAY, NOW NOBODY MOVE OR I'LL BURN THIS PLACE TO THE GROUND! *sirens, helicopter noises in distance* Oh, shit, they found where I am! *breaks and flees through window*

4 Conclusion

... Thanks, Tom. In other news, the Bank of SIGBOVIK was robbed by an armed gunman earlier today. Eyewitnesses report that the man was mumbling to an unknown target, attempting to pose as a paper submitted to a computer science journal. You don't see that everyday. Investigation is ongoing, and much like a crypto startup that'll probably go bankrupt in under a year, police still are desperately looking for the money. Police are also offering up to $5,000 for any information that could lead to an arrest; scan this QR code for more information.

Wearable RF-shield repositories

Abe Wits
Zurich
melon.mouse.games@gmail.com

J. Mijn
Bos en Lommer
/dev/null

ABSTRACT

We propose the use of wearable RF-shield repositories as a cost effective and realistic alternative to conventional protective measures in the fight against adversarial cables with RF capabilities.

1. INTRODUCTION

Radio Frequency (RF) transceivers (e.g. WiFi) can be hidden in the connector of a fully functional (USB) cable. Such a bugged cable can then be used for a range of attacks, including keylogging, injecting keystrokes and loading payloads on boot. Once a bugged device has found its way into an organization, it can be controlled remotely, up to ranges reaching kilometers. In 2008, this type of device was available to three letter agencies at a unit cost of 20k$ [5]. Today, they are available to anyone at a unit cost of around 100$ [6]. As a result, most organisations and valuable individuals should take defensive measures against bugged cables.

One state-of-the-art defence strategy is to ban or severely limit the use of cables (e.g. by cementing ports) in an organisation. However, this requires expensive modification of equipments, and complicates the use of hardware.

Another is to build a Faraday cage to shield an entire room or building [4]. This is even more expensive, limits the use of wireless networks, and ties equipment to dedicated physical locations.

A more flexible approach is careful supply-chain management. It is difficult and expensive to fully eliminate external access to company hardware. This problem can be partially alleviated by certifying cables. Certified cables can be recognized by labels or by using hard to mimic hardware [Fig 1]. This may deter an attacker unwilling to invest in faking certification. However, this defence strategy incurs significant costs and risks. When certified cables are unavailable, an employee may not be able to perform their job. Worse, an employee may be tempted to use an uncertified cable.

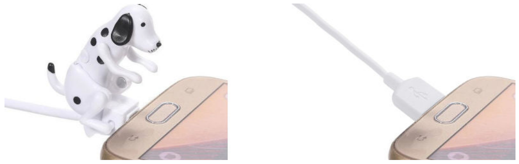


Figure 1: Hardware customization can ease the identification of a Certified USB cable (left) and makes a generic malicious USB cable (right) stand out. Source [3].

We propose a novel, inexpensive and practical strategy that allows for the use of uncertified cables, while providing full RF-cable attack immunity.

2. METHOD

RF-shield material can be molded around a cable to form a Faraday cage. We found that consumer-grade RF-shield material suffices, and should be preferred for financial and logistical reasons [2]. To allow the adhoc use of uncertified cables without a trip to the warehouse, a ready supply of RF-shield material should be carried by each employee in a portable repository. There are storage constraints - RF-shield material should not be crumpled or folded, as this could limit its efficacy. And organisational constraints exist - we rely on employees to follow protocol. To promote a positive culture and awareness around RF security, the RF-shield should be carried by employees in a visible manner. The visibility and storage constraints can be simultaneously satisfied by mandating that all employees carry a wearable RF-shield repository around the cranium. Material can be conveniently taken out of this repository at times of RF-shielding needs.

3. DISCUSSION

We recommend practicing placing RF-shield material around known RF-emitting devices before relying on our technique for mission-critical processes.

A wearable RF-shield repository has some side effects. Qualitative research has revealed that it may cause third parties to keep a distance. It appears to preemptively repel cyber security attacks and social interactions. We suspect this is due to third parties attributing strong cyber security skills to the individual wearing a RF-shield repository.

However, there are downsides to the visibility of the wearable RF-shield repository. An adversarial actor could deduce that you, being a visibly competent cyber security expert, are a worthwhile target for cyber espionage. We therefore recommend always combining a wearable RF-shield repository with anonymity enhancing apparel. The balaclava is effective and readily available, but illegal in several countries, including la France [1]. A more widely acceptable solution is the use of shades. A downside to them is that they potentially create an optical-reflection side-channel [Fig 2]. As a defensive strategy, we recommend the use of anti-reflective gadgets [Fig 3].

4. REFERENCES

[1] Assemblée Nationale, Projet de loi interdisant la dissimulation du visage dans l'espace public. `assemblee-nationale.fr/13/ta/ta0524.asp`, 2010.

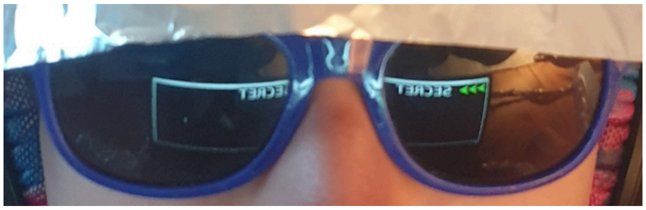


Figure 2: Notice the optical reflection sidechannel.

[2] EAFA. `www.alufoil.org/files/alufoil/trophies/2020/PressRelease/EAFA-Alufoil-Trophy-2020_Summary-PR_GB.pdf`.

[3] W. Ebshop. Certified Cable. `cafago.com/en/p-pa4185-1.html`.

[4] T. Liu and Y. Li. *Standard Study of Electromagnetic Information Leakage and Countermeasures*, pages 217–230. Springer Singapore, Singapore, 2019.

[5] NSA. Cottonmouth, NSA ANT. `en.wikipedia.org/wiki/NSA_ANT_catalog`, 2008.

[6] OMG. We are not including free ads for malicious tech in scientific work. Search for it yourself if you must., 2019.

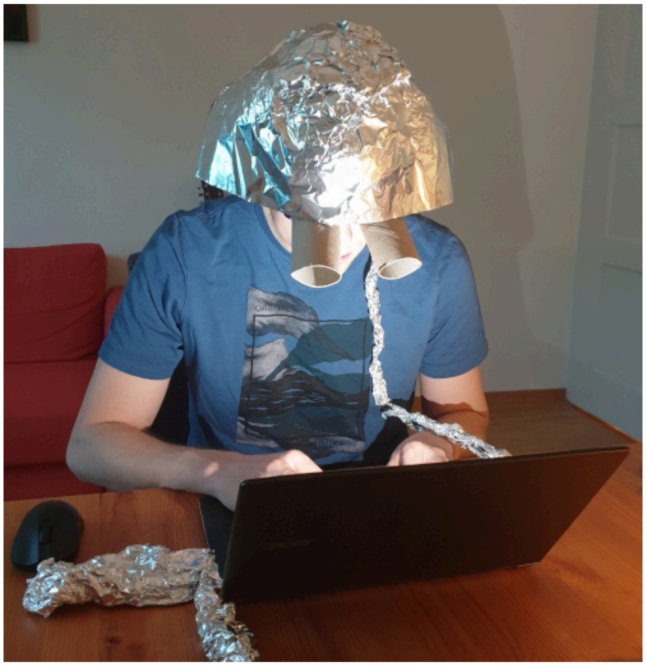


Figure 3: Wearable RF-shield repository, anonymity-enhancing apparel and anti-reflective gadgets.

Black Boxes

48 **A 23 MW data centre is all you need**

Samuel Albanie, Dylan Campbell and João Henriques

Keywords: machine learning, prophecy, Molly

49 **When Pull Comes To Shove... Do Both!**

Ullas A

Keywords: Parallel Programming, Graphs, Connected Components, GPUs

50 **man exorcism**

Keywords:exorcisms, unix, demonology

51 **2x a fake submission**

Tim Toady

Keywords: one letter keywords are not accepted: f, 21 is not a correct keyword: it contains no letters, =42 is not a correct keyword: it contains no letters

A 23 MW DATA CENTRE IS ALL YOU NEED

Samuel Albanie, Dylan Campbell, João F. Henriques
Pensive Prophets for Profit
Shelfanger, United Kingdom

ABSTRACT

The field of machine learning has achieved striking progress in recent years, witnessing breakthrough results on language modelling, protein folding and nitpickingly fine-grained dog breed classification. Some even succeeded at playing computer games and board games, a feat both of engineering and of setting their employers' expectations. The central contribution of this work is to carefully examine whether this progress, and technology more broadly, can be expected to continue indefinitely. Through a rigorous application of statistical theory and failure to extrapolate beyond the training data, we answer firmly in the *negative* and provide details: technology will peak at 3:07 am (BST) on 20th July, 2032. We then explore the implications of this finding, discovering that individuals awake at this ungodly hour with access to a sufficiently powerful computer possess an opportunity for myriad forms of long-term linguistic 'lock in'. All we need is a large ($\gg$ 1W) data centre to seize this pivotal moment. By setting our analogue alarm clocks, we propose a tractable algorithm[1] to ensure that, for the future of humanity, the British spelling of *colour* becomes the default spelling across more than 80% of the global word processing software market.[2]

> Where are they?
>
> Enrico Fermi, *CVPR 2022, virtual poster #65713*

1 INTRODUCTION

Accurate forecasts are valuable. From domains spanning battle outcomes (Babylonian soothsayers, 1900 BC) to precipitation nowcasting (Ravuri et al., 2021), humans have looked to hepatomancy and overly-cheerful weather presenters to assess the fate of their empire and to determine whether they need an anorak for their afternoon dog walk. Perhaps no topic has garnered more interest among professional forecasters than the future trajectory of technology (Lucian, 155 AD; Voltaire, 1762; Bush et al., 1945). However, future prediction is a difficult business, and the historical record of this discipline is somewhat patchy.[3]

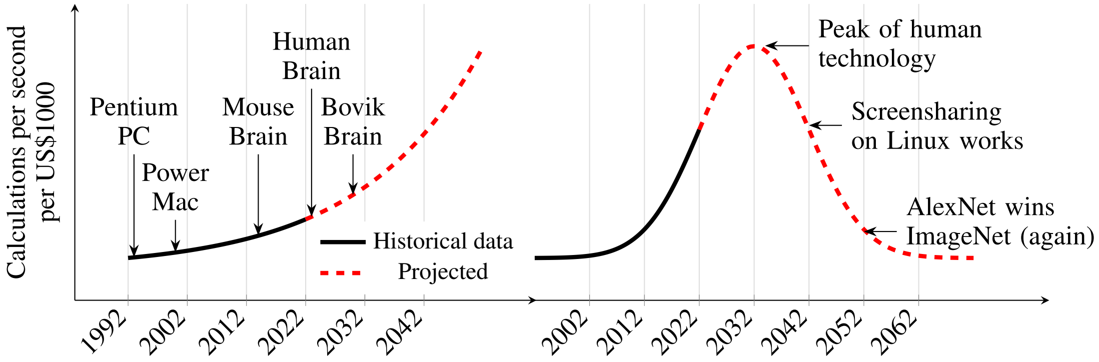
Science fiction authors have fared better at predicting the future, if you cherry-pick enough.[4] Rockets land themselves, some cars drive themselves (when someone is looking), and some humans think for themselves (and others). The opposing visions of Orwell (1949) and Huxley (1932) predicted two dystopias, one where people were controlled by fear and surveillance, another where they were controlled by endless entertainment and distraction. Rather than assess their fidelity, let us move swiftly on. Asimov (1951) proposed psychohistory as the science of predicting the future behaviour of human populations. By analogy to particles in a gas—assuming perfectly spherical humans in a

[1] Please see our Github repo (`https://github.com/albanie/A-23MW-data-centre-is-all-you-need`) for a permanent notice that code will be coming soon.

[2] We grudgingly acknowledge that `color` arguments in the matplotlib API should retain their u-free spelling for backwards compatibility. We are not complete barbarians.

[3] In addition to a widely publicised failure to predict the near-term feasible exploitation of atomic energy, Rutherford (1933) also failed to predict the global blue-black/white-gold dress debate of 2015.

[4] Even your first CIFAR-10 model was right 10% of the time.



(a) Kurzweil Curve

(b) Our Predicted Curve

Figure 1: **A principled approach to future extrapolation.** Standard approaches to future prediction, exemplified by the curve of Kurzweil (2005), are guided by empirical data and hardware trends. By contrast, our prediction relies on the tried and tested Central Limit Theorem. Note how ours is more symmetric and visually appealing. We note that the vertical[5] axis breaks down in the post-2032 regime, where US$1000 is increasingly meaningless. We therefore convert to the equivalent amount of pickled herring, and proceed. We will also forecast several events using our predictive model, which we now predict will be detailed in a later section of this article.

vacuum—while each human's state is unknowable, one can derive statistical quantities such as peer pressure, shear stress, and cringe factor.

To take on the challenge of future technology prediction, several themes have emerged in prior work. One body of research has leveraged historical trends and hardware laws with hints of exponential nominative determinism to underpin forecasts (Kurzweil, 1990), lightly handcuffed by physical limits (Bennett & Landauer, 1985). A second approach, pioneered by Gabor (1963), acknowledges the impossibility of future prediction and instead advocates inventing the future, or sagely producing smarter agents to take care of the inventing (Yudkowsky, 1996). However, empirical scaling laws lack a principled theoretical foundation, while actively inventing the future sounds exhausting, and honestly just waiting for Moore's law is much easier.

In this work we propose a third approach that both benefits from rigorous statistical theory and has a higher chance of completion within the modern 37.5 hour UK working week (including tea breaks). Our starting point was to turn to that reliable workhorse of modern statistical theory, the Central Limit Theorem. In brief, the Central Limit Theorem states that when random variables with sufficient independence of thought are summed, their normalised summation tends asymptotically towards a Gaussian distribution. We must, of course, address the delicate question of whether the Central Limit Theorem can legitimately be applied to our future forecasting problem. Thankfully, the Central Limit Theorem can and should be applied to all problems involving uncertainty, and few topics are as uncertain as the future.[6]

The technical foundations thus laid, the first key contribution of this work is to observe that recent decades of exponential growth in the number of transistors lovingly squeezed onto a microchip[7]

[5] According to Wikipedia, "the word 'vertical' corresponds to a graph as traditionally rendered on graph paper, where vertical is oriented toward the top of the page, regardless of whether the page itself—or screen in the computer era—is embedded upright or horizontally in physical 3-space. The 'top' of a page is itself a metaphor subordinate to the convention of text direction within the writing system employed. 'Vertical' has a concrete as opposed to metaphorical meaning within gravitational frames; when a page is held 'upright' in 3-space these two concepts align with the top of the page also gravitationally vertical. Horizontal is often equated with 'left' and 'right', but note that in the typographic convention of recto and verso, left and right also take on additional meanings of front and back" (Wikipedia, 2022).

[6] Historical misapplications of the Central Limit Theorem have arisen simply by applying the wrong variant of the Central Limit Theorem—there are a wonderful assortment of variants to choose from. Out of respect for the venerated theorem, we never acronymise.

[7] Some of which can be attributed to Tesco's convenient part-baked microchips, which can be finished in a home oven to a golden crisp.

neatly fits the steep climb of a hand-drawn bell curve (Fig. 1). A textbook application of the Central Limit Theorem then yields two striking insights. First, it enables us to compute with extremely high precision the date at which technological progress (as measured by transistor snugness or FLOPs per 1K USD) will peak: *3:07 am (BST) on 20th July, 2032*. Second, it enables dead certain modelling of uncertainty in our predictions, because the prediction itself is a Gaussian distribution (note that we carefully select our technology units such that the area under the curve sums to one). With the future now standing a little more naked in front of us, it behooves us to consider the questions motivated by this discovery, discussed next.

What is the cause of the decline? While prior work has explored the stasis-inducing potential of soma (Huxley, 1932) and drouds (Niven, 1969), we initially posited that widespread use of large language models for coding and paper writing will result in increasingly automated science production, with the associated atrophy of biological brains, and a drift of technological focus to matters of interest to disembodied machines.[8] However, our later findings suggest that a simple coupling of reward hacking with an ageing Flappy Bird clone will bring about the reversal.

What research opportunities does the decline present? Options abound for machine learning researchers who can acquire control of a 23 MW computer at 3:07 am. While some may choose to lay down an ImageNet top-1 SoTA that outlives Bradman's test career batting average, we propose instead to pursue a worthier cause. Leveraging a novel algorithm (described in Sec. 3.3), we plan to conduct a 51% attack on spellings of the word *colour* amongst the internet corpora used to train autocorrect systems. By doing so, we protect ourselves from annoying red-underlined squiggles well into our retirement, or worse, learning several new spellings.

Will we have to give up wireless headphones? Sadly. According to curve, by 4000 AD the last of the great homo sapiens engineers will strain to build an Antikythera orrery. It will probably be flat.

The remainder of this paper is structured as follows. In Sec. 2, we wilfully misrepresent prior work to increase our chances of acceptance at the prestigious SIGBOVIK conference. Next, in Sec. 3, we describe in more detail our mischievous methodology to topple the status quo among the spelling tsars. Finally, after minimal experimentation in Sec. 4, we conclude with conclusions in Sec. 5.

> Prediction is very difficult, especially if it's about the future.
>
> Niels Bohr

2 Uncomfortably Closely Related Work

Quo vadis? A short history of future prediction. Formative early work by Pythia (1400 BC) at Delphi showed the considerable value of goat sacrifice, noxious fumes and keeping just a hint of ambiguity when forecasting (revived recently as "confidence intervals"). In the more recent machine learning literature, creative researchers have sought to learn representations by predicting the near future in video (Ranzato et al., 2014; Vondrick et al., 2016). To the best of our knowledge, however, no such work has considered video extrapolation decades into the future, which would clearly be more useful. More related to our time scale, we note that the farsighted "standard run" *limits to growth* model of Meadows et al. (1972) for population growth adopts a similar "what goes up must come down" philosophical bent to our forecast, but their graph is spiritually closer to a Laplace distribution than a Gaussian and hence the Central Limit Theorem cannot be so confidently invoked.

Claims of universal sufficiency. Other than our own, the most notable past attempts to make gloriously broad claims about a framework's ability to fulfil all of a researcher's needs have considered Trust (Welter, 2012), A Good Init (Mishkin & Matas, 2016), Attention (Vaswani et al., 2017) and Love (Lennon et al., 1967). We note with consternation that, as a corollary of being all-encompassing, the above must be mutually exclusive. This unfortunate property is a product of clumsy first-order logic, and may be relaxed in the future by a reformulation using cold fuzzy logic, and warm fuzzy feelings. A cautionary note: not all prior literature has your best interests at heart.

[8] These include beating all other models on larger boards of multi-coloured Go, achieving 100% accuracy on MNIST, and a sequence of increasingly sophisticated puns about endianness.

Previous work has attempted to claim "all your base" (Toaplan & Namco, 1991) and employed the elegant tools of algebraic geometry to construct n-dimensional polytope schemes that efficiently separate you from your financial savings (Fouhey & Maturana, 2013).

> Who controls the past controls the future:
> who controls reddit spelling convention
> controls the past.
>
> George Orwell, *1984*

3 THE ROAD AHEAD

As noted with some trepidation in the introduction, the Central Limit Theorem assures us that the technological progress of humanity will soon stall before relinquishing its historical gains. In this section, we first explore possible causes of this trajectory (Sec. 3.1). We then fearfully investigate the implications of our findings (Sec. 3.2).

3.1 CAUSES

We initiated our analysis of plausible causes of technological decline by enumeration: a butterfly wing flap in downtown Tirana in 1658; Poincaré's premature death in 1912; the inexplicable density of Milton Keynes' roundabouts in 2022. Yet none of these seemed fully adequate.

The science fiction literary canon suggests an alternative cause. In desperate search of relief from Slack notifications, increasing numbers of technologists over the next decade will turn to *soma* (Huxley, 1932) and reasonably priced wireheading options (Niven, 1969), thereby functionally removing themselves from the innovation collective. However, although it is reasonable to expect a one-way loss of many great minds to these attractors (slowing the progress curve), our modelling suggests that a significant fraction of the engineering community have already submitted to the iron rule of *focus mode*. These hardy individuals are likely immune to such sirens, though they are hungry, owing to their missing two out of every three food deliveries and three out of three impromptu group trips to Nando's. We therefore sought to understand how the last bastion of resilient *focus moders* too could fall. To this end, we trained a one-and-a-half layer Transfomer (Vaswani et al., 2017) to predict future events from a series of carefully curated, temporally ordered Wikipedia entries.

By sampling from the transformer, we learnt that by the year 2031, humanity will have achieved a new *Gini coefficient* SoTA, wisely distributing 99.99999% of its total wealth among three living humans (each unreachable by Slack notifications) and an adorable Labrador named Molly. It is into such a society that on December 31st, 2031, an anonymous game designer[9] releases an adaptive, self-learning reboot of the 2013 mobile classic, *Flappy Bird* (FB). Designed to maximise user engagement, the FB algorithm soon begins to explore reward hacking strategies. Although it garners a following of just 17 users, its impact is monumental. Of these 17, three sit atop the Forbes "3 over 30 trillion USD" list, and each is incapacitated, unable to stop playing lest they lose their progress (which cannot be saved). Shortly thereafter, global capital flows grind to a halt and the silicon sandwich factories begin to fall silent.

In shock, the world turns to Molly. She would like to help if she could, but only after walkies, and she's not sure that she can remember her BTC passwoof [sic]. Starved of donations, Wikipedia servers are powered down, and the sole Stack Overflow answer explaining how to undo recent git commits is lost. Alas, any replications of this sacred knowledge were deleted long ago as duplicates by ModeratorMouse64. A period of mourning ensues. There is still hope. The situation could be salvaged. But it requires the technologists to speak to other humans. And so, because that would be awkward, the opportunity is lost.

[9] We can't be sure who. But if we had to guess, it would be SIGBOVIK legend, Dr Tom Murphy VII.

> Perfectly balanced, as all things should be.
>
> ImageNet-1K (2009)

3.2 Implications

We next consider implications. We begin by noting that modern "brain-scale" language models represent not only a great demo and a potential existential threat to humanity: they also open up a clear attack surface on the previously impregnable English spelling and grammar kingdom. The reason is simple. Just as doing away with cruft is a programmer's biggest joy, we ditch old headache-inducing paradigms with enthusiasm. As we transition from fast approximate string matching to language models all the way down, the battleground of canonical spelling moves from carefully curated small-scale corpora to vast, scarcely filtered swathes of the internet.

To exploit this observation, we propose an approach inspired by the 2007 run on the UK bank, Northern Rock, and its exemplary demonstration of a *positive feedback loop*. Noting that members of the celebrity GPT family of models (Radford et al., 2018; 2019; Brown et al., 2020) are trained via log-likelihood maximisation, our objective is to ensure that 51% of the training data adopts our preferred spelling conventions. To operationalise this plan, we propose HMGAN (Her Majesty's Generative Adversarial Network), a neural network architecture inspired by Goodfellow et al. (2014) that ingests sentences of written English and rephrases text to maximise a measure of similarity against a corpus of text respectfully inspired by blog posts penned by Queen Elizabeth II. Since future auto-correcting spelling assistants will likely derive from future generations of these models, and since human authors passionately hate both red squiggly lines beneath words and changing default settings, a simple majority in frequency counts across training corpora should suffice to ensure that future text converges asymptotically to our convention.

3.3 This spells trouble

In the tradition of machine learning papers, we have many biases and assumptions that underpin our choice of datasets and targets. Against tradition, we list some of ours here. Our canonical English has the following features.

1. U-philic: wherever there is an "or", there ought to be an "our", except where that isn't true. We support colour, valour, and flavour, but disown doour, wourk, and terrour.[10]
2. Zed-phobic: zed (or izzard) is supposed to be a shock.[11] Incurs a penalty in our loss function.
3. Ell-shippers: ells are meant to be together. It would be an unethical AI practice to separate them at this time.
4. Singular variants for all *pluralia tantum*: a trouser, a pant, a scissor, and a thank are all valid choices under our orthography.
5. Tildes: allowable in mathematics, and in ã (the authors declare no conflict of interest in this decision).

We seek to provide the tools for baking in the consensus mode, which we will be releasing open source, with the stipulations that they not be used by anyone seeking to promote 'color' over 'colour' or by *les immortels* of the Académie Française[12].

[10] No wonder we need an AI spell-checker, clearly these rules are not specifiable. That being said, two out of two linguists we interviewed suggested that computer scientists shouldn't be deciding these things.

[11] Oxford disagrees, and has a well-publicised infatuation with z that beggars belief. It is a rare beggar that believes an etymological ζ trumps a curvy French *s*.

[12] The astute reader may note that we nevertheless cherish our French linguistic influences, favouring the Anglo-French *colour* over the Latin *color*.

3.4 A critical point

To plan ahead, as required by our grant application, we must address two key questions. First, what magnitude of computing muscle is required to conduct a successful 51% spelling attack on the global internet? Second, how can we ensure that the spell checker trained post-attack achieves and maintains pole spell-check position on *paperswithcode*, thereby ensuring uptake and the positive feedback loop we seek to create?

Charting the future compute curve. Forecasting future computation is fraught with difficulty, but forecasting the power draw of future computers may be simpler, and thus we turn to this approach. Over the past decade, efficiency gains have ensured that increases in energy consumption across global data centres have been fairly modest, growing approximately 6% between 2010 and 2018, reaching 205 TWh (Masanet et al., 2020). Through a combination of tea-leaf interpretation and curve fitting, we estimate that global data centre energy usage will be in the region of 227 TWh in 2032.

Before turning to the implications of this estimate, let us note a few additional salient observations. First, thanks to healthy market competition in the services sector, it is likely that an increasing fraction of the world's computing budget will be allocated to the operation of friendly sales chatbots. By 2032, we believe that almost all written English content appearing on the internet will arise in unintentional bot-to-bot conversation. As such, the training corpora for future spell checkers will be curated almost entirely from their transcripts (after filtering out the phrase "I'm sorry, I didn't understand that. Did you mean 'How can I give you five stars on Amazon?"'). Second, note that approximately 0.01% of written English (Leech, 1992; Davies, 2010) corresponds to usage of the word 'colour' or 'color'—a frequency that we assume will be reflected in the chatbot discourse. Third, observe that the best spell checkers must keep themselves relevant by training on only the most recent linguistic usage (discussed in more detail below).

In light of the above, we see that a successful 51% attack to establish a simple majority spelling of 'colour' can be achieved by surpassing global chatbot text generation for a short period of time—just long enough for spell-checkers to fixate on the new spelling. By employing a text synthesis algorithm (HMGAN) whose energy consumption matches that used by chatbots, we find that a 23 MW data centre suffices for our aims (a derivation of this estimate can be found in Appendix A). Since the chatbots will, of course, rely on the latest spell-checkers to avoid embarrassing their corporate overlords, they will quickly transition to the new spelling. Then, as technology begins to decline, content production will drop, and spell-checkers will be forced to consider ever-expanding temporal windows to curate sufficient training data, rendering it ever more costly to reverse the trend.

A timeless spell-checker. If spell checkers are to keep up to date with modern argot (similar to, but decidedly not, a fungal disease), it is critical that they are trained on the most recent and most correct data. To this end, we propose a diachronic language model spell-checker. Extending the work of Loureiro et al. (2022)[13] to meet our needs, we commit to releasing a language model and spell-checker update every three hours, trained on a carefully curated and corrected stream of Twitter data. Our last update, for example, was trained on 123.86 million tweets, weighted according to the logarithm of the number of retweets and hearts, and with spelling errors corrected where appropriate. Importantly, our time-aware language model has knowledge of recent events and trends, allowing us to capture language as it is used in practise, not how the Académie Française ordains. For example, we observed a significant spike in the incidence of five-letter words, especially those with many vowels. Unlike existing language models, ours was successfully able to mirror this trend and dilate or contract words entered by our users to five letters. An unforeseen side-effect was the conversion of some words to coloured rectangles ⬜🟩🟨⬜🟨, but this is likely a consequence of our data augmentation strategy. It is crucial that all language-based tools be kept abreast of recent events and trends, because AI models of this sort deep freeze the cultural landscape from where the training data is obtained. It is highly unethical for AI researchers to participate in a system that creates cultural feedback loops and stagnation, over-privileging the status quo at the expense of the kids[14]. We further observe that Twitter is an excellent and unbiased source of international language usage that does not reflect any one cultural background, and so is a particularly good dataset for our purposes. It is also on the Académie's list of banned linguistic sources, which in our view speaks to its merits.

[13] An admirable instance of the "lour" convention.

[14] The spelling of colour is the only exception to this rule.

However, it is not enough to periodically release a language model fine-tuned on the last 3 hours of corrected Twitter data. In the fast-evolving world of language, this is already unusably out of date. Our previously-described model failed, for example, to autocorrect "vacation" to "staycation". It is incumbent on GPT-as-a-service (GaaS) providers to provide up-to-the-minute language models, motivating the development of temporally-predictive models. As we shall show in our experiments, our Predictive Diachronic Generative Pretrained Transformer (PDGPT) model effectively captures contemporary language usage, reflecting the most recent events, and is moreover able to generate geo- and time-localised results.

> You either die a grad student, or you live long enough to become R2.
>
> Dr. Harvey Dent (NeurIPS Area Chair), 2008

4 Experiments

In this section, we first validate our ideas in a simplified setting by considering 51% attacks in the context of the *British Bin Colouring Problem* (Sec. 4.1). We then compare our PDGPT spell-checker to the existing state-of-the-art (Sec. 4.2) and discuss civilisational impact (Sec. 4.3).

4.1 The British Bin Colouring Problem

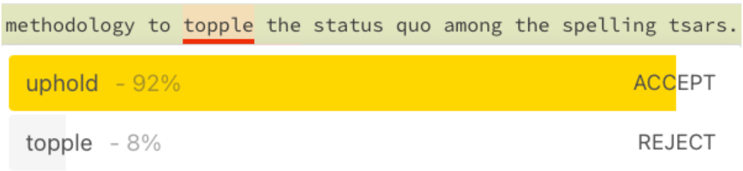
The *British Bin Colouring Problem* (BBCP) refers to a mathematical problem that is more practical than graph colouring and more general than bin packing. The task is as follows. On Wednesday evenings (or your local bin collection night), the objective is to wheel out the colour of bin that causes maximum mischief to your neighbours. Wheeled out bins of the wrong colour *will not be collected under any circumstances*. You have three choices: (1) black - unfiltered, (2) blue - recycling, (3) green - garden waste. Central to this problem is the assumption that, to avoid social tension, almost all neighbours will copy their neighbours' bin colour, rather than check the official bin collection colour through the local government website. Note, that you must account for upstanding citizens, who will put out the right bin colour regardless of their neighbours, misleading leaflets, or inaccurate local government websites. The problem is NP-Hard and environmentally significant.

We consider an instance of the BBCP for the residents of Grantchester, a picturesque village in Cambridgeshire. Our strategy was simple: we first employed HMGAN to craft a sequence of royal entreatments to wheel out the blue coloured bin on a green bin Wednesday, and sent leaflets to this effect at addresses generated via a Sobol sequence to ensure reasonable coverage. We then wheeled out our own blue bin and waited. A combination of stochastic upstanding citizen locations and wheel-out race conditions complicated our analysis, leaving us in some doubt as to whether a 51% bin colour majority would achieve our desired ends. To counter this intractability, we employed a systematic strategy of hoping it would work.

Unfortunately, the results of this experiment were unpromising. In our enthusiasm, we had failed to wait until 27th March, thereby missing the transition to Daylight saving time. As a consequence, it was too dark for our neighbours to determine our bin colour and were thus uninfluenced. They also did not take kindly to unsolicited leaflets, and are, by now, quite frankly tired of our shenanigans.

4.2 Simulated comparison to the state-of-the-art

Undeterred, we turn next to an evaluation of our PDGPT spell checker, capable of both autocorrection and event prediction. By backtesting on historical data, we find events and spellings successfully predicted or caused by our model include *quarantinis* but not *maskne*. More concerningly, despite our comprehensive set of three unit tests, PDGPT insists on auto-correcting our own use of 'colour' to 'color', undermining the core objective of our enterprise. This speaks to the formidable challenge of over-turning the spelling status quo (see Fig. 2), the difficulty of controlling large language models and the fact that we still don't really understand what the `.detach()` function does in PyTorch.



methodology to topple the status quo among the spelling tsars.

uphold - 92% ACCEPT

topple - 8% REJECT

Figure 2: **When it comes to spelling, it's not so easy to topple the status quo.** Increasingly dystopian modern grammar checkers, when applied to the close of the introduction of this article, let us know that we stand little chance of success. We soldier on.

Table 1: Masked token prediction for our Predictive Diachronic Generative Pretrained Transformer (PDGPT). For each three-hourly model, the table displays the top-3 predictions ranked by their prediction probability.

Models for 01/04/2022	I'm working from ⟨mask⟩.	I keep forgetting to bring a ⟨mask⟩.	Looking forward to watching ⟨mask⟩!
09:00 UTC	bed home afar	smile purpose baguette	closely yall snow
12:00 UTC	home upstairs tenerife	bag mug charger	snow skaters twitch
15:00 UTC	memory home work	charger friend bottle	tv bridgerton ash
18:00 UTC	shelter cover asgard	torch bottle party-hat	flames revelry ragnarok

In Tab. 1, we present qualitative results from our three-hourly predictive models trained for 01/04/2022[15]. Our model predicts the ⟨mask⟩ token in context, the same mode we use for text auto-completion. While we are not yet able to evaluate the quality of these predictions, we expect them to be rigorously validated by the time of publication. We note that our model has learned to reason about localised weather systems, plausibly predicting snow late in the season with no actual meteorologically-relevant input.

4.3 Limitations, Risks and Civilisational Impact

One limitation of our approach is reflected in our complete inability to produce convincing experimental results to date, even in Grantchester. We believe that this limitation will be overlooked by reviewers who recognise other merits to our work, such as our heavy use of footnotes which lend much needed academic gravitas to the text.

A risk of our approach is that it may encourage other researchers, notably our beloved American colleagues, to pursue a similar framework, escalating into a transatlantic arms race in which ever larger fractions of the planet's energy are dedicated to controlling spelling conventions.

In terms of civilisational impact, the stakes are as high as ever. John Wesley, the founder of Methodism, notably considered the removal of the *u* a 'fashionable impropriety' in 1791 (Mencken, 1923). But in 2032, for the first time the opportunity will exist for eternal spelling lock in for the large swathe individuals who don't remember to change the default setting on their spell-checker.

[15] We presume our model uses the DD/MM/YYYY convention.

5 Conclusion

We have presented a rigorous statistical analysis of historical and future computational trends and ascertain the date and time at which technology, on average, will peak. We have leavened this potential downer with an account of the implications of this finding and the concomitant opportunities that this presents. We provide the tools for myriad forms of long-term cultural and linguistic "lock in", with a particular focus on spelling and an especial concern for that of "colour". We expect this work and attitude to resonate throughout the following crepuscular decades as we revert to our respective agrarian utopias.

There are many promising avenues for future research. However, the relatively brief time before technological decline sets in precludes large-scale projects if significant computation is required by the work. One correspondent has suggested that a mixture-of-Gaussians model is more appropriate for our extrapolation, to better conform to conformal cyclic cosmology (Penrose, 2010), as all theories must. A mixture-of-infinite-Gaussians is intellectually appealing, but computationally infeasible (without using a RKHS, which is unfashionable). A more plausible direction is a new time and date scheme based on standard deviations from the Gaussian technology curve. Significant further research is required to quantify whether the two-to-one mapping from years to standard deviations will be problematic.

Acknowledgements. We acknowledge with gratitude our families, friends, colleagues, and cheerful international research community. We also acknowledge the profound wisdom of A. Sophia Koepke.

References

Isaac Asimov. Foundation. 1951. 1

Babylonian soothsayers. Clay tablet of a sheep's liver, . https://www.britishmuseum.org/collection/object/W_1889-0426-238, 1900 BC. 1

Charles H Bennett and Rolf Landauer. The fundamental physical limits of computation. *Scientific American*, 253(1):48–57, 1985. 1

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 3.2

Vannevar Bush et al. As we may think. *The atlantic monthly*, 176(1):101–108, 1945. 1

Mark Davies. The corpus of contemporary american english as the first reliable monitor corpus of english. *Literary and linguistic computing*, 25(4):447–464, 2010. 3.4, A

English-Corpora. https://www.english-corpora.org/, 2022. A

David F Fouhey and Daniel Maturana. On n-dimensional polytope schemes. *SIGBOVIK*, 2013. 2

Dennis Gabor. Inventing the future. 1963. 1

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 3.2

Aldous Huxley. *Brave new world*. 1932. 1, 1, 3.1

Ray Kurzweil. *The age of intelligent machines*, volume 580. MIT press Cambridge, 1990. 1

Ray Kurzweil. *The singularity is near: When humans transcend biology*. Penguin, 2005. 1

Geoffrey Leech. 100 million words of english: the british national corpus. *Language Research*, 28 (1):1–13, 1992. 3.4, A

John Lennon, Paul McCartney, George Harrison, and Ringo Starr. Love is all you need. *Society for Our World*, 1967. 2

Daniel Loureiro, Francesco Barbieri, Leonardo Neves, Luis Espinosa Anke, and Jose Camacho-Collados. Timelms: Diachronic language models from twitter, 2022. 3.4

Lucian. *A True Story*. 155 AD. 1

Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. Recalibrating global data center energy-use estimates. *Science*, 367(6481):984–986, 2020. 3.4, A

Donella H Meadows, Donella H Meadows, Jørgen Randers, and William W Behrens III. The limits to growth: a report to the club of rome (1972). *Google Scholar*, 91, 1972. 2

Henry Louis Mencken. *The American language: a preliminary inquiry into the development of English in the United States*. 1923. 4.3

Dmytro Mishkin and Jiri Matas. All you need is a good init. *ICLR*, 2016. 2

Larry Niven. *Death by ecstasy*. Galaxy Publishing Corp., 1969. 1, 3.1

George Orwell. Nineteen eighty-four. 1949. 1

Roger Penrose. *Cycles of time: an extraordinary new view of the universe*. Random House, 2010. 5

Pythia. The oracle at delphi. 1400 BC. 2

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018. 3.2

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 3.2

MarcAurelio Ranzato, Arthur Szlam, Joan Bruna, Michael Mathieu, Ronan Collobert, and Sumit Chopra. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014. 2

Suman Ravuri, Karel Lenc, Matthew Willson, Dmitry Kangin, Remi Lam, Piotr Mirowski, Megan Fitzsimons, Maria Athanassiadou, Sheleem Kashem, Sam Madge, et al. Skilful precipitation nowcasting using deep generative models of radar. *Nature*, 597(7878):672–677, 2021. 1

Ernest Rutherford. *Address at the British Association for the Advancement of Science*. 1933. Author note: we mischievously engage in the selective interpretation of the original quotation to support our story. Rutherford's statement was clearly not intended as a future prediction - "Anyone who says that with the means at present at our disposal and with our present knowledge we can utilize atomic energy is talking moonshine" - a claim that was arguably true until Leo Szilard's breakthrough hours later. 3

Toaplan and Namco. Zero wing. 1991. 2

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 2, 3.1

Voltaire. *Micromégas*. 1762. 1

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Anticipating visual representations from unlabeled video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 98–106, 2016. 2

W3Tech. Historical trends in the usage statistics of content languages for websites. `https://w3techs.com/technologies/history_overview/content_language`, 2021. A

Friederike Welter. All you need is trust? a critical review of the trust and entrepreneurship literature. *International Small Business Journal*, 30(3):193–212, 2012. 2

Wikipedia. Abscissa and ordinate — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Abscissa%20and%20ordinate&oldid=1066098077`, 2022. [Online; accessed 29-March-2022]. 5

Eliezer Yudkowsky. Staring into the singularity. 1996. 1

A Compute power estimates

In this appendix, we provide detailed calculations of our data centre size requirements. In the spirit of appendices, which are not written to be read, the derivation was carefully not checked for errors. Computations were performed on an iPhone with a slightly cracked screen.

Masanet et al. (2020) estimated global data centre energy consumption in 2018 at 205 TWh, corresponding to a rise of 6% since 2010. Extrapolation over 14 years at a growth of 6% every eight years yields an estimate of 227.007 TWh for 2032, or 25.914 GW average power. The present edition of the Corpus of Contemporary American English (Davies, 2010) indexed by English-Corpora (2022), finds 124814 occurrences of 'color' among 1 billion words (constituting 0.0124814% of all word usage). The British National Corpus Leech (1992) contains 11,135 counts of 'colour' among 100 million words (amounting to 0.011135% of all word usage). Averaging these terms, we determine that 0.0118082% of written English (at least, the kind that makes it into linguistic corpora) consists of some variant of the word colour. By 2032, we assert that all but a vanishing fraction of written text will consist of conversation between chatbots, which will dominate power consumption. Approximately 60.6% of internet content is written in English at present (W3Tech, 2021), and we assume the chatbots will follow this trend. We further assume, naturally, that the chatbots will use spell checkers, and will approximately maintain historical word frequency usage. Thus, it becomes clear that a successful 51% attack requires us to obtain a proportional share (by power) of global data centre compute that will outpace the use of colour in chatbot-to-chatbot conversation. This amounts to 1.91147 MW = $(26.712328\text{GW} \times 0.606 \times 0.0118082/100) \times 1000$. Thanks to the Matthew law, and the fact that spell checkers will be trained only on the most recent corpora to avoid staleness, we need only overwhelm the text generation of the chatbots for a very short period of time. However, note that our attack strategy requires us to insert instances of 'colour' into minimally valid sentences that will make it into the spell checker training data. On believe that pithy sentences with an average length of 11 words will suffice for this task, bringing our power needs for 22.93764 MW. Finally, we allocate a safety buffer of 0.01 MW capacity to install involuntary operating system updates forced upon us at 3:06 am, leading to a 22.94764 MW power profile.

When Pull Comes To Shove... Do Both!

Ullas 😄

Department of Computer Science and Automation,
Indian Institute of Science, Bangalore
email: ullas@iisc.ac.in

Abstract

Graph algorithms are ubiquitous. Wait, graphs are ubiquitous. Period. It is an axiom of life [1] that the more something is ubiquitous, the more people want more of it. Graphs are no exception. And Computer Scientists eat graphs for breakfast — and weighted graphs, let's not forget (and if you ask, "What do they eat for lunch and dinner, then?", well, they don't. Processing the breakfast takes them enough time to last until next day, because... BIG DATA!!). The apocalyptic scenario where graphs have become so pervasive that one needs to breathe graphs rather than Oxygen for survival is not too far away. Sooooooo, it helps to be prepared, you know...

DIALOGUE

Reader: Wait, how is that an abstract?

Author: Because it is anything but concrete.

Reader: What is this paper even trying to tell? What research problem does it address?

Author: All in its own good time.

Reader: And that is definitely NOT a valid reference that you have put there.

Author: It doesn't violate any definition of reference that is known in the literature.

Reader: Self-references don't count!

Author: It doesn't invalidate the veracity of the statement nevertheless.

Reader: This is not even the way to write a research paper, you know? Research has to be pathbreaking. It has to do something that nobody has embarked upon before, something preferably so mindblowing that— Ohhhh!

Author: Exactly! I am so glad you see what I have done.
Reader: Wait, why you are writing down my words?
Author: To make this dialogue a part of the paper itself.
Reader: GASP! I see myself here! Dialogues in a research paper! WHAT HAVE YOU DONE?

PREREQUISITES

You are a Computer Scientist, in any sense of the phrase, since the field has grown so vast that it has encroached upon adjacent fields, and the neighbours are threatening litigation.

POSTREQUISITES

You remain a Computer Scientist, haha.

1 INTRODUCTION

Imagine you are the ringleader of an organization that kills terrorists. So wait, doesn't that make you a terrorist as well, because you inspire terror into the hearts of terrorists, and isn't inspiring terror the very dictionary definition of a terrorist? Ok, I see that the first word of this section was a colossal mistake. Do not imagine too much.

Devil: So, by that definition, if you are killing terrorists, shouldn't you eventually commit suicide?
Author: So much for the phrase "An idle mind is a devil's workshop".
Devil: Ok ok, now I get it — why you have asked the reader to imagine. So as to keep me at bay.
Reader: Um..., a Devil in a Research paper?
Author: Well, we do need someone to play the Devil's Advocate.
Reader: And a dialogue in the middle of the paper too?
Author: Hey, you are featuring in it as well. No complaining.

Let us return, refreshed, from that delightful and much-needed digression, in light of the heavyweight machinery that awaits. As a ringleader, what is your primary goal?
(A) To buy a ring for yourself (to lend credibility to your title)
(B) To try to remain in one piece by the end of the operation
(C) To publish research papers on graphs
(D) To, um, kill terrorists?

[The above MCQ is left an exercise to the alert reader]

Reader: I thought you said we had returned from the digression.
Author: We digressed again. As we will throughout. For that matter, any non-digression is a digression.

Your skills are so much in demand that you spend three-fourths of the year in fixing and replacing your door... This is owing to the fact that people come breaking down your door, demanding for your skills, probably because they have dispensed with all the necessary politeness of knocking on the door first because you are so much in demand, for the few seconds it takes them to knock might mean that you are already taken by their competitors.

Anyway, your task is to identify the terrorist networks so that you can be sure that once you locate and capture one terrorist in that network, that terrorist will lead to others in the network, who can in turn lead to others they know, and so on (it is inherently assumed that your interrogatory prowess involves the most advanced methods at your disposal, owing to which, the captured terrorists WILL certainly reveal all they know, out of sheer, er, terror at the thought of what you are capable of doing to them if they chose not to sing — which really *really* REALLY begs the question of who is the worse terrorist over here, really). Your repute as a ringleader of your organization is well known, to the point that every time you go out on your missions, dogs move out of your way. And any stray dog lingering on your path is pulled back by the other dogs who realize that you mean business. And in case you are wondering where dogs come in to the scene, aren't the terrorists themselves dogs of a kind, and the worst kind at that (the word *kind* may be a little too kind to be used on them)? Ok, such is your repute that nobody would date you out of sheer fear. In fact, you probably come from an era before the concept of dating was invented. I guess that makes you a "Predator", haha!

Anyway, this is the idea of a Connected Component in a Graph (er... no, not the dogs), where the nodes are the terrorists, an undirected edge connects two nodes if the two terrorists belong to the same organization and know each other (as with most terrorist organizations, not everyone will know everyone in it, but it is guaranteed that from any one of them in the organization, you can reach anyone else (apologies if that sounds too ominous)), and all the terrorists within a network form a single "connected component", because they are all connected (blood is thicker than water, and all that, you know), while those of other terrorist networks form their own components, disparate from each other (it is well known that terrorists, unlike in academia, do NOT collaborate — their Erdos numbers are laughable).

RIDDLE

Why are the nodes of a graph so stressed?
Because they are living on the edge.

The problem then, is simple. You need to identify and label each node in the graph with the component it belongs to.

Reader: Couldn't this one line alone appear in the abstract instead of all this rigmarole? It's precise. It would have saved so much time.

Author: Yes. And you can go visit Agra and come back home without seeing the Taj Mahal. It's precise. It would save you so much time.

DISCLAIMER

This paper discusses the Connected Components Graph Algorithm, and readers familiar with it can skip the rest of the paper, or read it for fun.

META-DISCLAIMER

The above disclaimer should really really have appeared in, like, page 1, possibly even before the abstract, and not this late in the paper.

META-META-DISCLAIMER

In the event that the disclaimer really did appear as suggested, the necessity for the above meta-disclaimer would have been entirely obviated, and hence it would have been absent.

META-DISCLAIMER

Oh yeah? Then in that case, the Meta-Meta-Disclaimer would have been absent as well. Take that, you nosy brat of a meta-meta-disclaimer!

META-DISCLAIMER:

META-META-DISCLAIMER:

HEY! You are encroaching upon my space!

You don't deserve a box of your own. And I will teach you to call me a brat!

Now I guess you merging into my box should make us a single connected component.

Hey, and who on earth are you down there?

AUTHOR:

Now now guys, this has gone too far.

[redacted]

The reader is strongly urged to desist from asking what happened within the above box. Let the black hole in the page above be a void of no return to the imagination.

Anyway, as it so happens, the definition given above is an operational definition, and so the algorithm is evident to the most myopic bat—

Devil: Wait, aren't bats blind by default? Isn't a myopic bat a genetic improvement upon the species?

Having lost the beautiful metaphor to the Devil's Domain, we will proceed with a succinct description of the obvious algorithm to find Connected Components in a Graph.

Algorithm 1

```
Input: Terrorist Graph
Output: Terrorist Graph with labelled Components

function begin()
{
    Give_Up();
}
function begin_again()
{
    while nodes remain in the Terrorist graph:
      Let t be a terrorist; visit t, add t to VisitedList, giving t a ComponentId.
      while VisitedList is not empty:
          Capture a terrorist from VisitedList and add him to CapturedList,
          giving him same ComponentId.
          ***************************
          For each contact c given above, pay a "friendly" visit, adding c to
          VisitedList.
      Increment ComponentId.
    return home
}
```

The above is the standard algorithm in the literature — ok ok, I think the word "standard" might have raised some eyebrows, and might have tested the elasticity of the word, but that can be easily remedied by massaging the eyebrows in order to lower them. Questions are awaited.

Reader: What is the point of function begin?

Author: To make a first attempt at solving the problem, but giving up owing to fear of the repercussions. Therefore, begin_again is entered into after much convincing. This is typical of all hard problems. They go by the common name of “Not Programming”, from the very words of dismissal that programmers use when asked to program such problems. Often abbreviated to NP or NP-Hard.

Reader: Are you serious?

Author: Haha, no no of course not! But this makes for a more interesting definition of NP and NP-Hard Problems than what is defined in the literature.

Reader: What are those asterisks in Line 5?

Author: The unprintable means that you use to extract the information from the captured terrorist. The asterisks hide the gory details, leaving the means used to the reader’s imagination. It is a blocking statement, because it completes only after the list of the terrorist’s known contacts are given, leading to Line 6.

Reader: Why the VisitedList and CapturedList? Why two lists? Why not just one?

Author: You can revisit a terrorist but capture him only once. It is assumed that captured terrorists do not have the means to make an escape.

Reader: Why would you want to revisit a terrorist?

Author: To get more information.

Reader: Ok. All clear.

In summary, the above algorithm is a straightforward way to start from a given terrorist and follow through all of his known contacts, tarring them all with the same brush, and once this terrorist’s network is fully explored, moving on to a disparate and disconnected terrorist and repeating, with a different brush, the process all over again (because your organization’s name is: *“Kabhi AlQaeda Na Kehna”*).

Problem solved. End of pap— WAIT!! The order in which the terrorists are visited follows either Depth-First Search or Breadth-First Search as popularized in the literature. However, both of these are dimensional travesties upon considerations of efficiency. Therefore, we shall lift ourselves from the three-dimensional shackles that confine us and move into the fourth dimension by aiming for Time-First Search instead... You are supposed to be the ringleader of an ORGANIZATION. And unless, by the word “organization”, you mean the arrangement of your own body organs, it is likely that there are going to be others working with you. So if you alone are out hunting, what do your minions do? You don’t want them idle, if only because they need to be wary of the Devil.

Devil: Hey, did someone invite me?

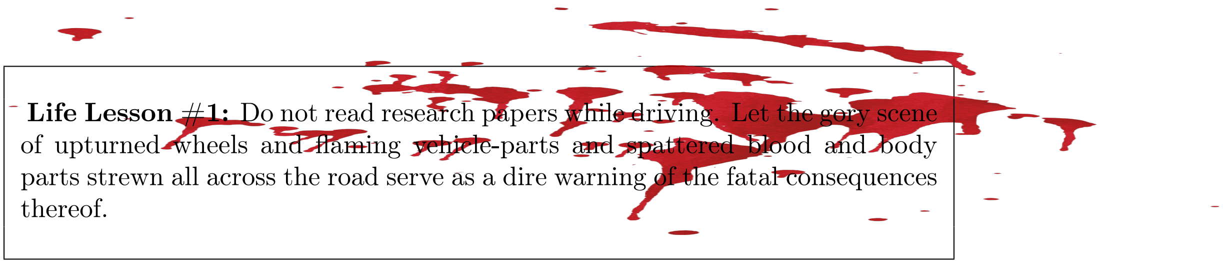
Author: Speak of the Devil!

Devil: And the Devil comes! So, if you don’t use your minions, then you will be killing all the terrorists yourself? That would make you a serial killer!

And thus, the only way to avoid being labelled for life as a serial killer is to use your minions for you, and thereby be labelled a parallel killer instead.

Enter Parallelism!

SCRRRREEEEEEEEECCCCCCHHHHHHHHHH!!!
CRAAAAAAAAASSHHHHH!!!



Life Lesson #1: Do not read research papers while driving. Let the gory scene of upturned wheels and flaming vehicle-parts and spattered blood and body parts strewn all across the road serve as a dire warning of the fatal consequences thereof.

Having witnessed at first hand the untimely demise of a precious reader, lost to surprise twists of a research paper, a stark reminder of how brutal academia can be, and having observed a minute's silence to commemorate the departed soul, let us now proceed to Section 2 with a cautious step, an acknowledgement of the dangers that embellish our path ahead.

2 MILPSRLAEAL NE SMSAE

I apologize for the title of this section. It's supposed to be "PARALLELISM EN MASSE", but well, this only just goes to show the inherent dangers of unconstrained parallelism. You attempt parallelism in displaying characters, and this is the result. You probably didn't use any locks. But then, if you use too many locks, it would hurt performance severely. Which I guess means that you need to use just the right number of locks. This, dear Readers, is called the GoldiLOCKS zone, haha.

Given the fact that you have many agents in your organization, how should you tackle the task of identifying the connected components in the terrorist graphs? And to make it more simple, let us assume that we already have a list of the names of the terrorists, we just don't know who belongs to which organization (We shall leave the aftermath of identifying this as an exercise in imagination, and concern ourselves with just identifying the connected components). And you have such a large number of employees that you don't even need to go admire the night sky for the stars — you only need to look at your organization from far away, and you can make out shapes of constellations owing to so many of the employees.

Devil: Speaking of constellations, aren't constellations really just another glorified name for connected components of a graph?

What you'd like, what you'd really like is, well, world peace, but since that's a lofty goal, let's aim for something more achievable. So what you'd like is to give one single algorithm for all your employees to follow. What you would NOT want is to give each employee a different algorithm, since that would be a nightmare for you to keep track of who's given what task.

Very crudely, this is the idea of a GPU (Graphics Processing Unit) — a processor that has massive parallelism in the form of "threads" that all execute the same instruction at a time.

Devil: You know you are lying, right? By omission.

Author: Sigh ok ok. To come entirely clean, not all threads are executing the same instruction. Threads are grouped into "warps" and all threads in a warp are executing the same instruction. But those of a different warp need not. But all are MOST DEFINITELY executing the same program — they just need not be at the same line in the program. And furthermore, warps are grouped into thread blocks. It is not ok to launch thread blocks with insufficient work to do, because they would then die off quickly, and you see... Block Lives Matter.

Devil: Groan! So much for pedantry. And puns.

What might, at first glance, seem like an evident and trivial algorithm, is, upon deep reflection (and refraction and diffraction and interference and other optical phenomena as well) not really, and the reader is therefore urged to desist from prematurely presenting an obviously incorrect algorithm.

Reader: Well, it *is* trivial. Just assign each minion to one terrorist in the graph. And each minion visits that node and labels it with a unique component id, and follows that terrorist's contacts, and labels them with the same component id.

Author: You have already been warned that this is not a trivial algorithm. So if we use your approach, when a minion follows a terrorist's contacts, he will reach other terrorists, right?

Reader: Yeah.

Author: Who have already been assigned to some other minions, who all followed the same algorithm, and have already labelled those terrorists with their own component ids.

Reader:

Author:

Reader:

Author:

Reader:

Author:

The above speechlessness proves the statement made earlier that this algorithm is not trivial. After a long enough interval of open-mouthed shock on the part of the reader, an interval long enough to accumulate flies in the reader's mouth, necessitating drastic measures to purge them thereof, let us proceed to elicit further suggestions.

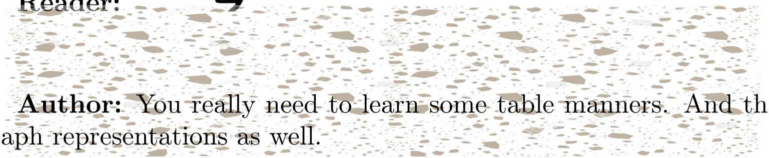
Reader: Ok, how about the following? Same as before — each minion gets one terrorist. But now, when that minion reaches the terrorist, maybe he paints the terrorist with some unique colour to act as component id. Then follows the contacts of this terrorist. If he sees an uncoloured contact, he goes ahead and colours that contact because clearly this contact belongs to the same component. And keeps following that contact's contacts and so on, until all nodes are coloured.

Author: And what if this minion finds a contact that has already been coloured by some other minion?

Reader: That means he has already been visited by some other minion, and the two minions really are exploring the same component. So one of the minions has to go back and recolour his terrorist with the other minion's colour, to show that there is only one component.

Author: And suppose the other minion also stumbles upon the first minion's terrorist, and sees him coloured a different colour, indicating that this terrorist was reached by someone else, so he goes back and recolours his original terrorist with the other minion's colour. And we are back to square one, or circle one, if you will, because two contacts are coloured differently, when they should be coloured the same.

Reader:



Author: You really need to learn some table manners. And that includes graph representations as well.

Reader:

Author:

Reader:

Author:

Despite our repeated attempts at solving the problem, it appears as if some malevolent cosmic entity is operating behind the scenes to ensure failure with high probability, where high probability can be defined as—

Devil: Hey, did you call me?

Author: Oh damn! I was hoping that not using your name would help.

Devil: I go by many names. I am aliased. Just because nobody ever chants my *Sahasranama*, doesn't mean it doesn't exist.

Author: Exorcise yourself!

In the interests of preserving the reader's sanity and time, and preventing the reader from expending an inordinate amount of energy, losing excessive mass in the process (well, owing to Einstein's famous equation) in an attempt to solve the problem, the solution is presented herewith.

Without loss of generality, but with also no real gain of specificity, but with sufficient loss of interesting and vivid descriptions of a colourful nature, let us assume that the terrorists (or nodes in the graph) are labelled from 0 to n-1 (it also helps from a standard fantasy trope point of view, because according to lore, if you give a thing a name, you give it power, and hence, let us not vest too much power into the terrorists, and divest them of their names). Start off by assuming that each terrorist has their component id to be the same as their label. By starting off assuming that each terrorist is operating solo, the goal is to refine this hypothesis to identify who else they are collaborating with, and eventually identify the component id for each terrorist. Each minion is requested to execute the following algorithm:

Algorithm 2: Let's call it... um... PUSH?

Input: Terrorist graph with each node labelled uniquely with its node-id
Output: Terrorist graph with each node labelled with its component id, where component id is the minimum node-id among all the nodes in that component.

```
function CC()
{
   If terrorist is painted black, go back home. Forget the rest of this function.
   Paint the terrorist black.
   Let L be the component id of this terrorist.
   ***************** (You know what to do)
   Visit each contact c obtained in the previous step, and do:
      If component id of c is more than L, set it to L *** (and
      paint terrorist c white).
}
```

In case you are wondering what the CC stands for (apart from unity, integrity and all that), it is Cafe Coffee Day who has realized that its Days are over and it might as well indulge its remaining life in Graph algorithms and in particular, Connected Components, because you see, a lot can happen over Graphs. The beauty of the above function lies in the fact that it is the same function that can be asked to be carried out by every minion. After all, let us not forget that you are the RINGleader, and hence you would like... ONE RING TO RULE THEM ALL!

ASSUMPTIONS

1. The terrorists are docile enough to not put up a fight and get themselves painted as per the whims of the minion.

2. What are the terrorists doing after you paint them? Yeah, you got that right — they are sitting right there, waiting for you to come back again and check on them to see if they have changed their colours. Of course, they are not just twiddling their thumbs — they are praying for an asteroid to come and deliver them from their miseries (despite many of the terrorists not really believing in, or for that matter, even knowing about, asteroids), or failing that, they are willing to settle for an Amazon agent, given that they seem to be so good at making deliveries.

RIDDLE

Where do terrorists hang out?
The gallows.

With the pride of accomplishment akin to parental pride evident in the public display of their newborn child, let us climb the nearest tallest rooftop in the vicinity and shout out for the whole world to see and bestow their admiration and envy upon our worthy newborn baby algori— oopsie...

AAAAAAAAAAAAAAAAAAAAAAAHHHHHHHHHHHHHHHHH!
CRRRAAAAAAAAASSSSSSSSHHHHHH!

Life Lesson #2: Do not climb rooftops with babies in the hope of public display. Unless the baby in question is (a) Spiderman, or (b) aerodynamically prolific, or (c) having high coefficient of elasticity. Babies are inclusive of newfound algorithms as well.

Let us once again mourn the passing away of yet another reader (infant readers count). It is evident that the number of readers at the beginning of the paper is most certainly not going to equal that which remains at the end, but let us plod on, hoping that no more fatal casualties ensue. Academia can be brutal. And it is best that the reader is made aware of this fact upfront. Academia is brutal because it stabs you in the back, just like Brutus, after whom the word “brutal” was coined, since that was what Brutus was known for, and everything else about him is forgotten. Brutus would seriously have blossomed and flourished in academia without breaking a sweat. And then one fine day when nobody was expecting it, he would have gone and joined the industry, having stabbed academia in the back.

Algorithm 2 has the advantage that it can be executed by all minions independently, reducing the overall runtime, so that each minion can go home for dinner (if not be dinner themselves at the hands of the terrorist they capture, but Assumption 1 provides them a protective shield of defense).

The reader’s teetering on the precipice of enquiry is manifestly discernible, and the reader is therefore encouraged to trip and give in to the temptation to satisfy curiosity.

Reader: But, but... I don’t understand this whole thing. I am not able to pinpoint what, but something seems kind of off.

Author: That’s because the algorithm presented above is incomplete, and I haven’t told you the whole story, haha! I wanted to see the reaction on your face.

Reader:

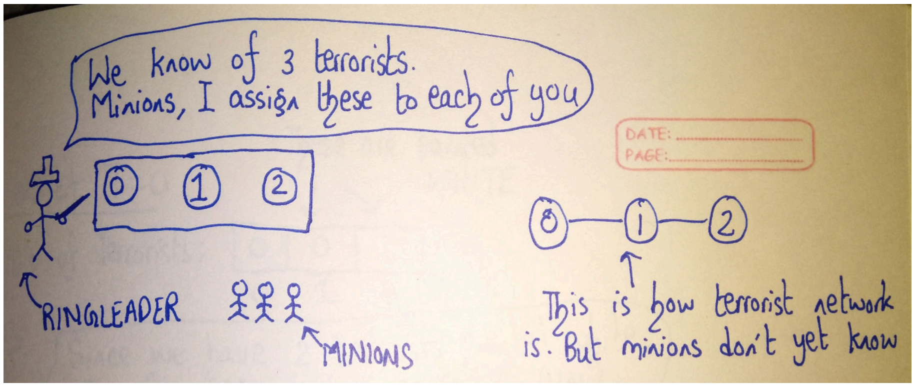
Life Lesson #3: Do not read research papers while eating or drinking.

Let us shed copious tears at the devastating and unprecedented loss of yet another callous reader to the brutal and unforgiving jaws of academia and research. Having attended the last rites and disposed of the remains of the eviden— er, I mean, the departed reader, let us plod staunchly ahead with a word of caution and warning to the remaining readers (if any) that by proceeding beyond the following point, they consent to do so at their own risk.

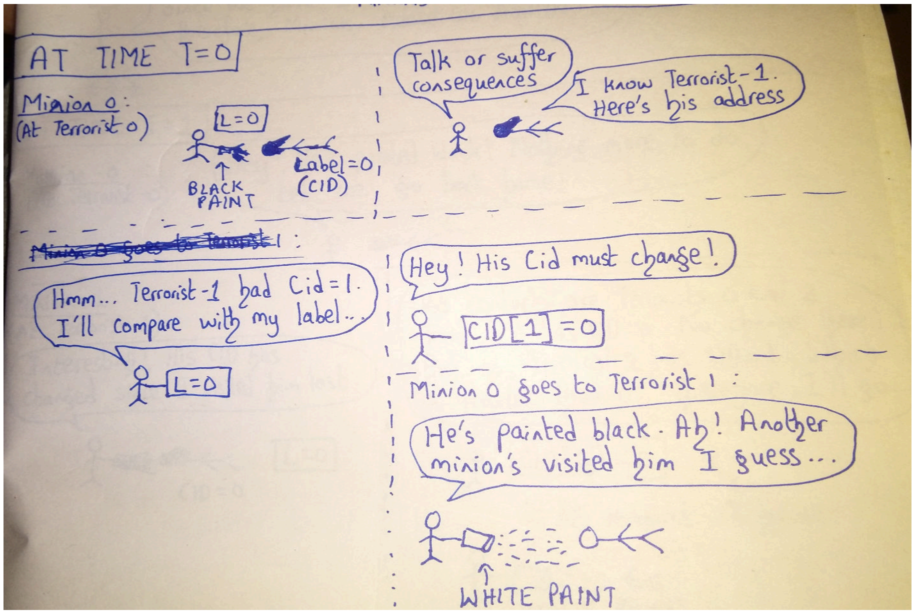
```
Algorithm 2 (The Rest Of It)
function RingLeader()
{
    Assign one minion each to each terrorist in the graph.
    As long as there is even one terrorist who is not painted black:
        Call CC()
}
```

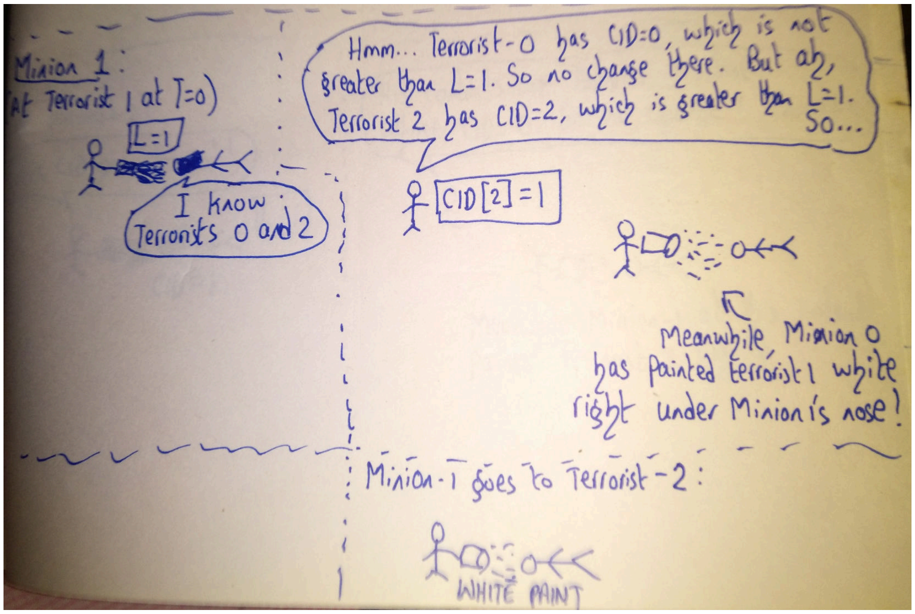
Out of essential courtesy, let us pause to allow the reader to get dazzled at the sheer effulgence and brilliance of this simple algorithm. Sunglasses are offered to protect the reader's eyes from the radiant glow emitted by this algorithm's awesomeness. It is a shameful pity that the first seven wonders of the world are monuments, and to remedy this egregious oversight of leaving out algorithms, it will be agreed that the above algorithm deserves an honoured place in the list. Dear Reader, say Hello to the Eighth Wonder of the World.

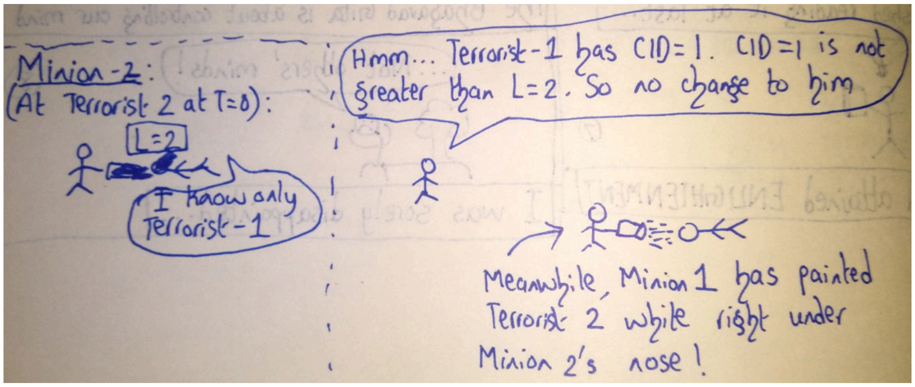
Reader: Um, Hello? But I am not sure I understand it still.
Author: Ah, that's because the effulgence of the program is making you close your eyes to the beauty of it.
Reader: Could you explain it?
Author: Let me show you instead.
Reader: Huh? Show?
Author: Come, Harry. Into the Pensieve.
Reader: Harry? Pensieve?[1] Wait, what is happe—AAAAAAAAAHHHHHHHH!

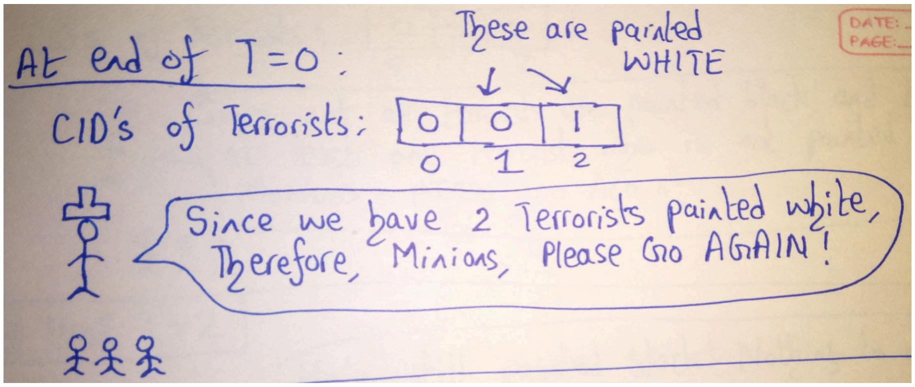

We know of 3 terrorists.
Minions, I assign these to each of you
0 1 2
DATE:
PAGE:
0 1 2
RINGLEADER
MINIONS
This is how terrorist network is. But minions don't yet know

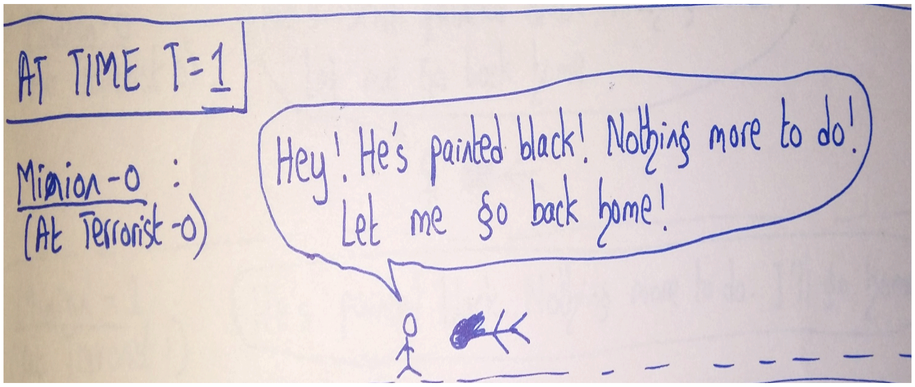
[1] A Pensieve is a magical device to store memories in the Harry Potter series [6]. Let us observe 2 minutes of mournful silence for those readers who have wilfully handicapped their own childhood by not reading the Harry Potter series.

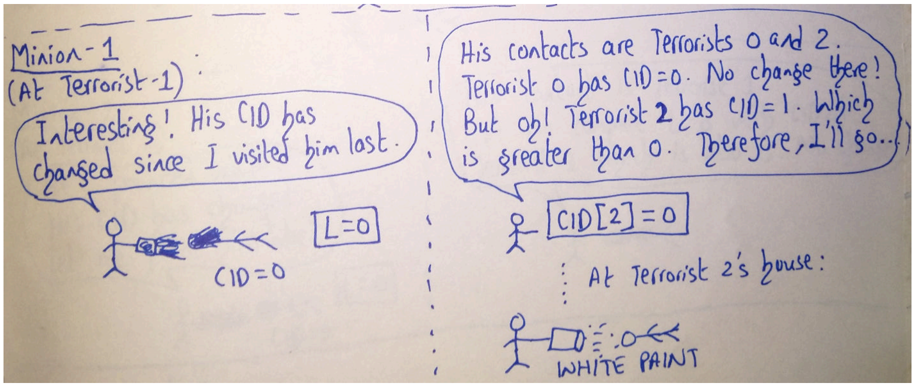

AT TIME T=0
Minion 0:
(At Terrorist 0)
L=0
BLACK PAINT
Label=0 (CID)
Talk or suffer consequences
I know Terrorist-1. Here's his address
~~Minion 0 goes to Terrorist~~ 1:
Hmm... Terrorist-1 had Cid=1. I'll compare with my label...
L=0
Hey! His Cid must change!
CID[1]=0
Minion 0 goes to Terrorist 1:
He's painted black. Ah! Another minion's visited him I guess...
WHITE PAINT

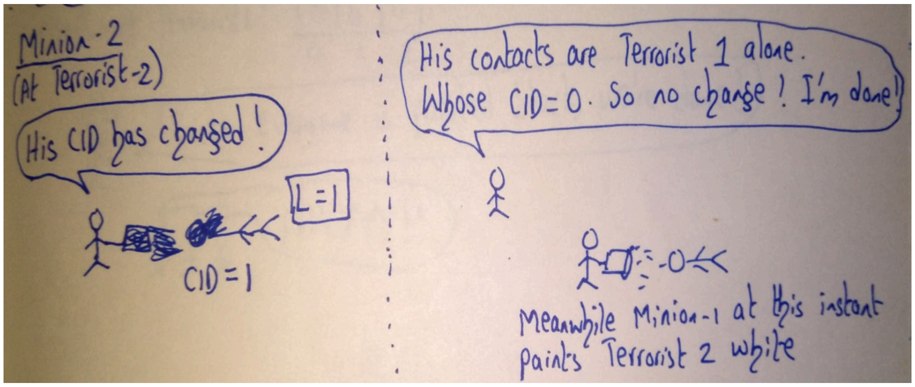

Minion 1:
(At Terrorist 1 at T=0)
L=1
I know Terrorists 0 and 2
Hmm... Terrorist-0 has CID=0, which is not greater than L=1. So no change there. But ah, Terrorist 2 has CID=2, which is greater than L=1. So...
CID[2]=1
Meanwhile, Minion 0 has painted terrorist 1 white right under Minion 1's nose!
Minion-1 goes to Terrorist-2:
WHITE PAINT

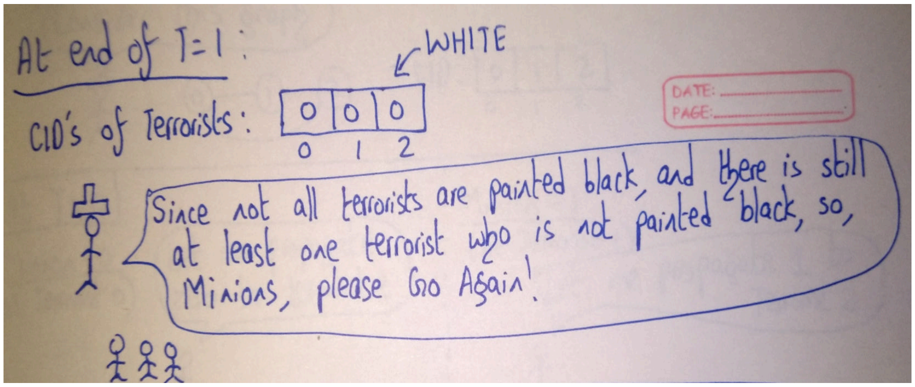

Minion-2:
(At Terrorist 2 at T=0):
L=2
I know only Terrorist-1
Hmm... Terrorist-1 has CID=1. CID=1 is not greater than L=2. So no change to him
Meanwhile, Minion 1 has painted Terrorist 2 white right under Minion 2's nose!

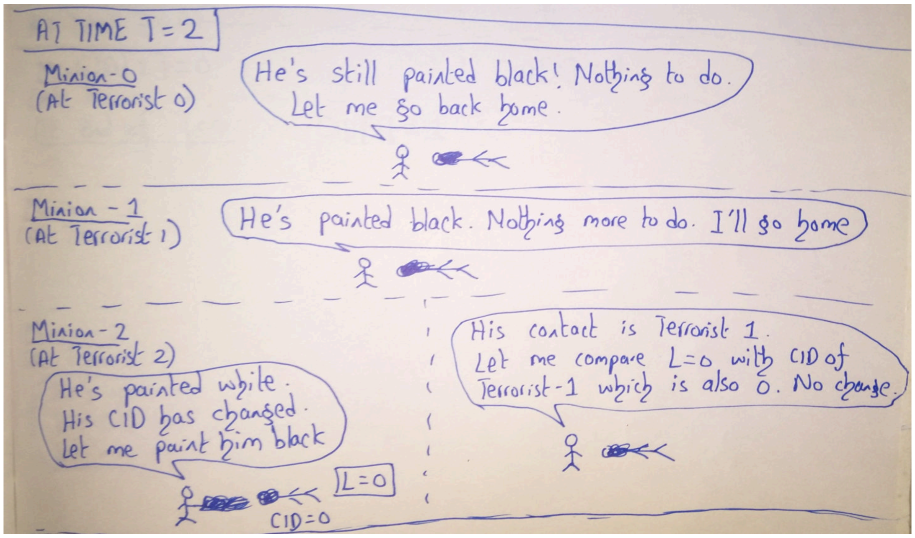

At end of T=0:
These are painted WHITE
CID's of Terrorists: 0 0 1
0 1 2
Since we have 2 Terrorists painted white, Therefore, Minions, Please Go AGAIN!
DATE:
PAGE:

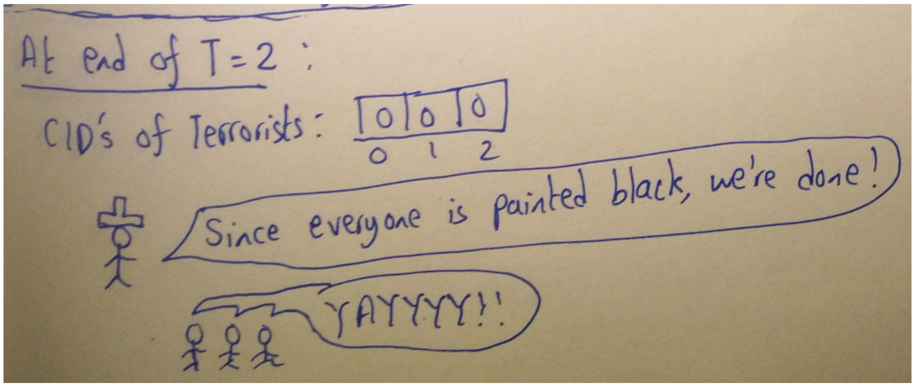

AT TIME T=1
Minion-0:
(At Terrorist-0)
Hey! He's painted black! Nothing more to do! Let me go back home!


Minion-1
(At Terrorist-1):
Interesting! His CID has changed since I visited him last.
L=0
CID=0
His contacts are Terrorists 0 and 2. Terrorist 0 has CID=0. No change there! But oh! Terrorist 2 has CID=1. Which is greater than 0. Therefore, I'll go...!
CID[2]=0
At Terrorist 2's house:
WHITE PAINT


Minion-2
(At Terrorist-2)
His CID has changed!
L=1
CID=1
His contacts are Terrorist 1 alone. Whose CID=0. So no change! I'm done!
Meanwhile Minion-1 at this instant paints Terrorist 2 white


At end of T=1:
WHITE
CID's of Terrorists:
0 0 0
0 1 2
Since not all terrorists are painted black, and there is still at least one terrorist who is not painted black, so, Minions, please Go Again!
DATE:
PAGE:


AT TIME T=2
Minion-0
(At Terrorist 0)
He's still painted black! Nothing to do. Let me go back home.
Minion - 1
(At Terrorist 1)
He's painted black. Nothing more to do. I'll go home
Minion-2
(At Terrorist 2)
He's painted white. His CID has changed. Let me paint him black
L=0
CID=0
His contact is Terrorist 1. Let me compare L=0 with CID of Terrorist-1 which is also 0. No change.


At end of T=2 :
CID's of Terrorists: 0 0 0
0 1 2
Since everyone is painted black, we're done!
YAYYYY!!

In summary, the Ringleader (the CPU) launches the operation (kernel) assigning minions (GPU threads) to each terrorist (node) in the graph. Each terrorist propagates his component id to his neighbours, so that any neighbours who have higher component ids update their component id so that within each connected component, each node will have the id of the lowest numbered node in that component. And when a terrorist's component id gets updated, that terrorist in turn must propagate this update to his neighbours, and thus, this is indicated by painting the terrorist white. A white terrorist indicates that in a subsequent iteration, his value needs to be propagated, whereas a black terrorist (apologies if this sounds overtly racist) indicates that his component id need not be propagated. Thus the algorithm terminates only when all terrorists have been painted black.

Alert Reader: I observed that when one of the minions is updating the component ids, another minion enters the terrorist's house right under the former's nose and paints the terrorist white.

Author: That's a good observation. This is because the CC function is executed independently by each minion. So while one minion is executing one part of the function, another can be executing another part of it.

Alert Reader: Also, I feel there could be an optimization. You see, for instance at time T=0, the CID of Terrorist 1 is updated to 0 by Minion 0, but at the same time, Minion 1 notes the CID value into L as 1 and not zero. In other words, the change made by Minion 0 is not reflected to Minion 1.

Author: Good observation. This is because we have seen a serialized version where we saw Minion 0's operation before Minion 1's operation. But remember that each of them are running in parallel. So it is quite possible that Minion 1 executes before Minion 0. Hence we store the CID value into a temporary variable L, and any changes made to CID are only stored locally by each minion, who then goes back home at the end of that time step (iteration) and writes the changed value to a global place.

AMBUSH!

AAAAAAAAHHHHH!!!

Life lesson #4: The above is to teach the reader that it is important to always be cautious while reading papers, because one never knows what is waiting around the corner.

Of course, since the goal of any research paper is to prepare the reader for life, and given that life is full of surprises, the above has been a step in the right direction, and must be applauded. Having given the readers sufficient time to catch their breath, let us proceed with the dialogue...

Alert Reader: We can optimize this, can't we? Maybe let each minion write the CID update onto the terrorist itself! So that when any subsequent minion comes there, the changed value will be propagated within that iteration, thereby reducing the number of iterations.

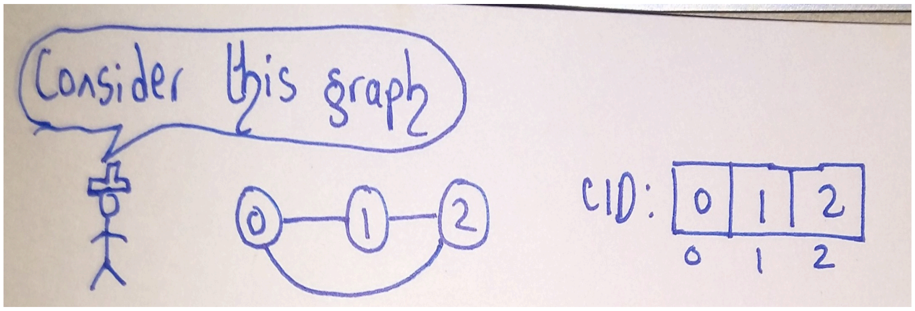
Author: Good. If we were to do that, such an algorithm would be called Asynchronous in the literature. What we have looked at is only the Synchronous case.

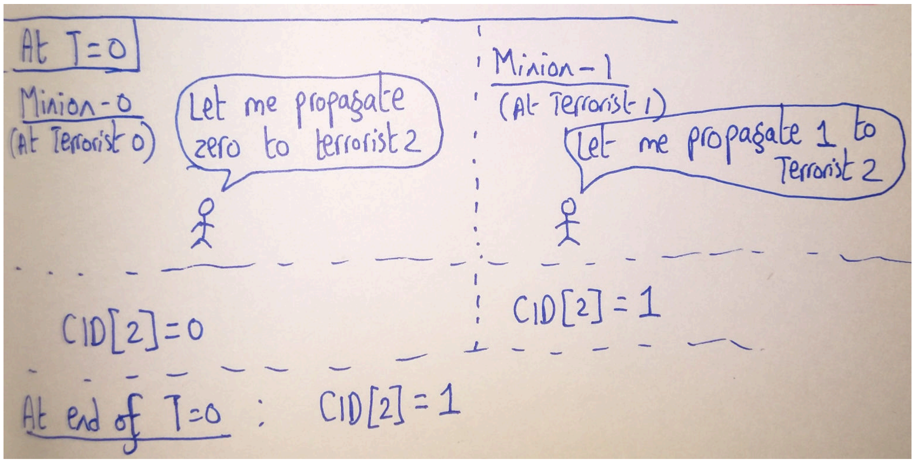
Alert Reader: I wondered about those asterisks in Algorithm 2 in the CC function. The last line of that function. Is that yet another place of third degree treatment?

Author: Excellent question. Those asterisks indicate a footnote to the alert reader. And since I am talking to one right now, let me expand on it. It indicates that that line is to be done atomically.

Alert Reader: Atomically? You mean, you go down to the level of atoms and change the positions and momentums of its electrons?

Author: Oh no no. Once again, let me show, rather than tell. Let us enter the subjunctive world where the operation was not atomic.




At T=0
Minion-0 (At Terrorist 0)
Let me propagate zero to terrorist 2
Minion-1 (At Terrorist 1)
Let me propagate 1 to Terrorist 2
CID[2]=0
CID[2]=1
At end of T=0 : CID[2]=1

Author: And now, at this point, Terrorist 2 is stuck with the value 1, and the update by minion 0 is lost. This would have been solved if we insisted that each minion gets exclusive access to the terrorist while comparing the cid value as well as updating it. Then, even if Minion 1 managed to write 1 to CID[2], Minion 0 would have seen that 1 at the time of its comparison and would have managed to overwrite that 1 with a zero eventually. Not like above where each of the minions could read the value of CID[2] at the same time and update it at the same time.

Alert Reader: Ok, but what happens if we try to break up the atomic operation?

Author: Don't do it.

Alert Reader: Eh, how bad can it get? A minion can just interrupt—

Life Lesson #5: Atomic energy is dangerous. Splitting the Atomic can be as catastrophic as splitting the Atom. Respect Atomic Energy.

Alert Reader: Great. So we have solved the problem at last. How elegant! We can all go back home now.

Author: Alas. What if I tell you we have only just begun?

HONK! HONK! HOOOONNNNNNNNNNNKKKKKKKK!!!!!

SCREEEEEEEEEECCCCCHHHHHHHH!!!

CRRRRAAAAAAAASSSSSSSHHHHH!

++reader_toll

Life-Lesson #6: Do not read research papers while crossing the road. Look up from your research papers, look to the left, then to the right, cross the road after verifying that it is safe to do so, and then resume reading the research paper.

Aspiring researchers, please take note of the above, lest we have expiring researchers.

3 EXPANDING THE PROBLEM

It is a truth universally acknowledged, that a single problem, in possession of a good solution, must be in want of a modification of the problem statement. Apologies to Jane Austen who is rolling in her grave at seeing her renowned sentence being abused in the above manner. Alas, she probably had too much Pride in her sentences, and too much Prejudice against Connected Components Problems.

But leaving aside Jane Austen's Senses and Sensibilities, could you think of how you could modify the algorithm yet maintain its correctness?

Devil: If you can't Push a terrorist too much, do you just Pull along?

Let us extend the most heartfelt gratitude to the Devil for edging us closer to our goal.

Reader: Ok. I have an idea. When we are comparing a terrorist's cid with that of his contacts, we propagate the source terrorist's cid to his contacts if any of his contacts c have a higher cid than the source. But what if we do the opposite? What if we update the source terrorist's cid if any of his contacts has a lower cid than the source?

No, dear Reader, this is not the end of the paper already. This blank page is a refreshment break, a much needed feature of any research paper that has the reader glued to its pages. Readers are requested to look up from the paper, stare at the horizons for a while, let out their breaths which they had held for such a long time, and after verifying that the pulse rate and breathing rate are normal, turn over to the next page and resume reading.

Reader: Did we reach the end of the paper already?

Author: No, that blankness was the glare of the sheer brightness of your idea which affected the ocular power of the unsuspecting reader. This, dear Reader, is the Pull variant of the algorithm as opposed to the Push variant we saw before.

Algorithm 3: PULL VARIANT

```
function CC()
{
    Let L be the component id of this terrorist.
    ***************** (You know what to do)
    Visit each contact c obtained in the previous step, and find the minimum
    cid among all c's. Call this min.
    If min < L, update Cid of this terrorist to min and paint him white.
}

function RingLeader()
{
    Assign one minion each to each terrorist in the graph.
    As long as there is even one terrorist who is not painted black:
        Paint all terrorists black.
        Call CC()
}
```

It is immediately apparent that there is some asymmetry between the push and the pull variants, especially when it comes to painting the terrorists. It is not a completely black and white matter, haha. In an attempt to prevent dizziness on the part of the reader owing to multiple forays into the pensieve, we shall not have another excursion therein, but leave the details to be worked out by the astute reader.

DISEMBODIED VOICE

Why aren't minions going back home in CC function if terrorist is already painted black? Why aren't minions painting terrorist black? What would happen if we follow very similar pattern of the Push variant and let the minions paint the terrorist black within CC, and not let Ringleader paint the terrorists black?

It is a basic premise, implicitly acknowledged in every fantasy, that one is obligated to listen to disembodied voices (if for no other reason that failure to do so might result in their voice becoming the next disembodied voice for others to hear), for they hold the key to many mysterious doors that remain otherwise stubbornly closed. Since this research paper is most unanimously agreed to be categorized under the fantasy genre, it behoves the reader to diligently attempt to follow the mysterious hints posed by the disembodied voice above, for this holds the answer to true understanding.

A detailed answer to all the above questions were given in the appendix, but the paper had to undergo an emergency appendicitis operation, and hence, its appendix is no longer available. The inconvenience is deeply regretted.

Reader: So which of the two versions is the better one?

Devil: Of course, the Push variant.

Reader: No, I think the Pull version is better.

Devil: Push.

Reader: Pull.

Author: How about we don the hat of a researcher and experimentally verify for ourselves? Isn't that much better than pushing around answers down our throats? Or pulling answers from a magic hat?

When you apply the tried and tested principles of inferring from data, you are abiding by the scientific spirit that provides everlasting salvation to its adherents. You might have to cross uncharted mountains, you might have to swim through shark-infested seas, you might have to face fire-breathing dragons, you might have to battle light-saber wielding aliens dropping from the sky through UFOs, but you can rest assured that the conclusions you reach are backed by hard solid data, and are not diaphanous wispy hypotheses drawn from a hat. You will attain the forbidden fruit of knowledge that can only be attained by the scientific spirit. You will then thank those mountains, seas, sharks, dragons, and aliens even if they are incapable of gracefully acknowledging your gratitude (though they might appreciate a bite from the forbidden fruit you are holding in your hand as you are thanking them).

Having gained sufficient experience with torturing confessions out of terrorists, let us now attempt to torture confessions from data, if only because it provides a welcome change from the monotony (although often enough dealing with data can cause far more terror than dealing with terrorists).

Type of Graph	**|V|**	**|E|**	**Push-CC-Time**	**Pull-CC-Time**	**Push-Iter**	**Pull-Iter**
gemsec-FB/artist	50515	819090	6.5194ms	9.4974ms	9	8
gemsec-FB/athletes	13866	86811	2.0312ms	2.7543ms	8	8
gemsec-FB/company	14113	52126	1.4808ms	1.8047ms	11	11
gemsec-FB/government	7057	89429	2.5921ms	3.6418ms	7	7
gemsec-FB/new_sites	27917	205964	3.4133ms	5.2294ms	11	10
gemsec-FB/politician	5908	41706	1.6289ms	2.3955ms	11	10
gemsec-FB/public_figure	11565	67038	1.8143ms	2.4578ms	11	10
gemsec-FB/tvshow	3892	17239	1.1815ms	1.3383ms	14	13
ca-CondMat	108300	93439	1.5367ms	2.2929ms	10	10
com-youtube	1157828	2987624	263.57ms	882.53ms	10	11
musae_git	37700	289003	143.44ms	159.07ms	9	7
roadNet-CA	1971281	2766607	92.329ms	79.105ms	554	542
Random	30K	10K	218.32us	196.08us	15	15
Random	100K	10K	109.33us	94.488us	7	7
Random	1M	1025	110.52us	123.25us	2	3
Random	1M	10K	198.13us	177.20us	3	4
Random	1M	100K	406.20us	359.71us	6	6
Clique	100	4950	309.57us	202.73us	2	2
Clique	1000	499500	10.896ms	2.2744ms	2	2
Clique	10K	49995000	179.02ms	92.475ms	2	2
Alternate-Edge	100	50	21.438us	16.638us	2	2
Alternate-Edge	1000	500	19.102us	16.703us	2	2
Alternate-Edge	10K	5000	31.965us	38.077us	2	2
Alternate-Edge	100K	50K	34.653us	24.542us	2	2
Alternate-Edge	1M	500K	165.71us	101.21us	2	2
Alternate-Edge	10M	5M	1.4220ms	824.05us	2	2
Alternate-Edge	50M	2.5M	24.180ms	4.7364ms	2	2
Alternate-Edge	100M	50M	27.113ms	29.344ms	2	2
Alternate-Edge	500M	250M	🐯	🐯	🐯	🐯
Alternate-Edge	1B	500M	🐯	🐯	🐯	🐯
Chain	100	99	941.57us	965.53us	100	100
Chain	1000	999	10.060ms	9.9702ms	1000	1000
Chain	10K	9999	111.12ms	102.14ms	10000	10000
Chain	100K	99999	1.48802s	1.47641s	100000	99986
RMAT	1M	1M	3.8789ms	5.7600ms	8	8

It is a well-known fact that any form of tabular data is completely glossed over by readers, and hence to avoid this indifference, two rows have been populated with dead cats, in the hope that they shall draw the reader's attention, and make them at least give a perfunctory glance at the table, to see if the other rows could hold any clue to the cause of their gruesome deaths. The dead cats are an omen. This provides a very chilling picture, and for this reason, it is very strongly recommended that this picture be placed inside multi-core processors so as to aid in the cooling process of the overheated system. Well, the dead cats in this case are an indication that flaming balls of hydrogen undergoing combustion reactions shine in the firmament, revolving in silent cosmological anticipation for the program to terminate and produce outputs for these inputs [2].

As with all important questions in life that attempt to ascertain the relative superiority between multiple competing alternatives, the answer is inevitably,

[2] Program? Where did that come from, you wonder? Well, for the interested reader, CUDA programs have been written to run the above algorithms, and the times shown above are the results of this program running on an NVIDIA DGX1 Server. More details are skipped in the interests of retaining the readers' sanity and interest. If you are the kind of reader who is interested to know more about those details, you are probably reading the wrong paper

“It depends” (or if not that, then surely it is “None of the above”). So much for the scientific method. It is evident from the above that there really isn’t any clear winner (although it can be ascertained that the number of iterations of the Pull approach never exceeds that of the Push). Therefore, let us resort to the age-old, tried and well-tested principle, when faced with a dilemma of choosing among n alternatives, to introduce an $(n+1)^{th}$ alternative, so that the original problem has been completely obviated (of course, to be supplanted by another problem, but let’s first bask in the rapture of annihilating the original problem, and pat ourselves on the back for such a commendable idea).

4 THE BEST OF BOTH WORLDS

Having basked in the ephemeral pleasure of having gotten rid of the original problem, long enough to realize that we have introduced another alternative to choose from, let us come to the sober realization that a problem (if not *the* problem) still exists, and we have a long way yet to go.

Reader: But what is the third alternative that you speak of?
Author: An amalgamation of the first two presented before!
Reader: Huh?

And before the reader even has time to draw his breath from the surprise of having received such an answer, or to realize what is just coming his way, we have—

BANG!!!!

Algorithm 4: HYBRID APPROACH

```
function CC()
{
    Wake up at 3 am, having tossed and turned on your bed for hours, unable
    to sleep.
    Question your career choices at having to go finding terrorists.
    Transition to questioning your life choices.
    Realize that you are duty-bound to identify the Connected Components
    of the terrorists' networks.
    Scream at the fates that have conspired to put you in this position.
    While there are no more neighbours attempting to batter down your door:
        Pacify the neighbours that your guttural screaming was nothing to
        worry about, and was an umm..., hehe... er... yeah, a car backfiring,
        right.
    Finally, get ready for your manhunting sojourn.
    Where the mind is without fear, and the head is held high...
        Realize that the previous statement wasn't really an algorithmic
        instruction in the truest sense of the phrase, but still... Your thoughts,
        your rules, or whatever...
        Get pushed out of your house to find the terrorists, and get pulled by
        your own moral and philosophical dilemmas.
        Realize that you have been both "pushed" and "pulled" in the previous
        step, and call this the "Hybrid Approach".
        THE END
}
```

Reader: WHAT ON EARTH WAS THAT?
Author: A much needed break from the intensity of the paper.
Reader: Um... So, there is no Hybrid Algorithm then?

Having given the reader enough breathing time to unwind, let us now present:

Algorithm 5: HYBRID APPROACH (the real one, because the previous one was just Pseudocode, living up to its name and being false code)

INPUT: Terrorist graph with each node labelled uniquely with its node-id.
OUTPUT: Terrorist graph with each node labelled with its component id, where component id is the minimum node-id among all the nodes in that component.

```
function CC()
{
    Let L be the component id of this terrorist.
    ***************** (You know what to do)
    Visit each contact c obtained in the previous step, and find the minimum cid among all c's. Call this min.
    If $min < L$, update Cid of this terrorist to min and paint him white.
    Visit all contacts c of this terrorist and if their Cid's are more than min, update them to min, painting them white.
}

function RingLeader()
{
    Assign one minion each to each terrorist in the graph.
    As long as there is even one terrorist who is not painted black:
        Paint all terrorists black.
        Call CC().
}
```

Why would the above algorithm, which is seemingly more complicated than either the Push or the Pull, and moreover, does more work than each of them individually, seem to work better? The key insight lies in the fact that the Hybrid approach propagates the component id across a node and all its neighbours in one iteration, unlike the Push or the Pull approaches which propagate the minimum cid across multiple iterations. It is surmised that this will lead to a reduced number of iterations and hence a reduced execution time (sorry for the phrase, for this has nothing to do with actual terrorist execution, which is a separate matter altogether).

Devil: But couldn't it be possible that though the number of iterations have reduced, each iteration does more work, and hence the actual time increases?

Questions from the Devil typically require careful thought and any flippant answers must be avoided at all costs. Being fully cognizant of the fact that no amount of sacrificial offerings in the form of kittens, or poor innocent beasts or even human beings shall placate the Devil, let us therefore seek our refuge with Data, to save us from eternal perdition at the hands of the Devil.

Type of Graph	|V|	|E|	Push-CC-Time	Pull-CC-Time	Hybrid-CC-Time
gemsec-FB/artist	50515	819090	6.5194ms	9.4974ms	36.821ms
gemsec-FB/athletes	13866	86811	2.0312ms	2.7543ms	3.4300ms
gemsec-FB/company	14113	52126	1.4808ms	1.8047ms	1.9672ms
gemsec-FB/government	7057	89429	2.5921ms	3.6418ms	4.8743ms
gemsec-FB/new_sites	27917	205964	3.4133ms	5.2294ms	7.0720ms
gemsec-FB/politician	5908	41706	1.6289ms	2.3955ms	2.8489ms
gemsec-FB/public_figure	11565	67038	1.8143ms	2.4578ms	2.8717ms
gemsec-FB/tvshow	3892	17239	1.1815ms	1.3383ms	1.5726ms
ca-CondMat	108300	93439	1.5367ms	2.2929ms	2.1538ms
com-youtube	1157828	2987624	263.57ms	882.53ms	960.59ms
musae_git	37700	289003	143.44ms	159.07ms	247.81ms
roadNet-CA	1971281	2766607	92.329ms	79.105ms	92.766ms
Random	30K	10K	218.32us	196.08us	179.12us
Random	100K	10K	109.33us	94.488us	94.968us
Random	1M	1025	110.52us	123.25us	140.98us
Random	1M	10K	198.13us	177.20us	175.92us
Random	1M	100K	406.20us	359.71us	380.03us
Clique	100	4950	309.57us	202.73us	319.46us
Clique	1000	499500	10.896ms	2.2744ms	3.9708ms
Clique	10K	49995000	179.02ms	92.475ms	156.06ms
Alternate-Edge	100	50	21.438us	16.638us	18.335us
Alternate-Edge	1000	500	19.102us	16.703us	17.887us
Alternate-Edge	10K	5000	31.965us	38.077us	20.638us
Alternate-Edge	100K	50K	34.653us	24.542us	35.260us
Alternate-Edge	1M	500K	165.71us	101.21us	164.02us
Alternate-Edge	10M	5M	1.4220ms	824.05us	1.4191ms
Alternate-Edge	50M	2.5M	24.180ms	4.7364ms	20.782ms
Alternate-Edge	100M	50M	27.113ms	29.344ms	50.301ms
Alternate-Edge	500M	250M	🐯	🐯	🐯
Alternate-Edge	1B	500M	🐯	🐯	🐯
Chain	100	99	941.57us	965.53us	629.93us
Chain	1000	999	10.060ms	9.9702ms	6.6927ms
Chain	10K	9999	111.12ms	102.14ms	64.185ms
Chain	100K	99999	1.48802s	1.47641s	1.16323s
RMAT	1M	1M	3.8789ms	5.7600ms	5.5498ms

The Data has betrayed us. All readers have been dragged into the darkest crevices of Hell with us. The increment operation on the reader_toll counter has been applied so much that the reader_toll counter itself has overflowed. And apart from this, there are still dead cats in those two rows. For all intents and purposes, the program has gone into a coma, refusing to listen to the tear-soaked pleas of its bedside relatives to wake up, or failing that, at least smile so that they'll know it's alright.

Moral of the story

Large inputs can cause unprepared programs to go comatose.

We are in a poor state, and readers who have been cursing the Author that this is what comes out of involving Devils in a self-respecting research paper must take solace in the fact that all is not yet lost. Because one must accept the fact that if we are being roasted alive in hell and are feeling the pain, then at least our nervous system is working fine, to be able to feel the pain, and hence all is not yet lost— QED! Kudos, Central Nervous System! Kudos!

Ok ok, that only seemed to have put oil to the fire.

RIDDLE

Q: How do you save one from being roasted in hell?
(a) Apply burnol
(b) Freeze hell over
(c) Exorcise the Devil
(d) Invoke the blessings of God

God: Say, did someone mention my name?

Author: Hallelujah! We have been saved! Save our Souls, God! SOS!

God: Um... you know what? Aren't souls supposed to be, like, you know, immortal or whatever? So they don't need to be saved really, do they?

Author: Then SOB God, SOB!

God: What! Are you using a swear word against me?

Author: No no, God! I meant, Save our Bodies!

God: Oh well alright. Then, tell me, why are you only looking at the time for CC? Shouldn't you be considering the overall execution time?

And that my dear Readers (or whatever charred remains of Readers there are), is the answer to the Riddle posed above. It is none of the given options (as should the answer to any self-respecting MCQ be). How do you save one from being roasted in hell? Well, by not letting them get into Hell in the first place, of course! Let us retract our steps and consider now the overall execution times, instead of merely the CC time.

The following table shows the overall runtimes of each of the inputs graphs for all the three variants, and like the previous table, for each input graph, the minimum execution time is highlighted in yellow. Oh, and by the way, if you're curious about the input graphs, the first few (until roadNet-CA) are standard real world benchmark graphs taken from the SNAP dataset [14], and the rest are synthetically generated graphs.

Type of Graph	|V|	|E|	Push-Time (s)	Pull-Time (s)	Hybrid-Time (s)	Hybrid-Iter
gemsec-FB/artist	50515	819090	1.1653	1.064	0.982	5
gemsec-FB/athletes	13866	86811	1.0826	0.8613	1.0323	5
gemsec-FB/company	14113	52126	0.979	1.0093	1.0853	6
gemsec-FB/government	7057	89429	1.0876	1.2233	0.8433	5
gemsec-FB/new_sites	27917	205964	1.0263	1.061	0.9446	7
gemsec-FB/politician	5908	41706	0.911	1.1803	1.0193	6
gemsec-FB/public_figure	11565	67038	1.26	0.8656	0.7556	6
gemsec-FB/tvshow	3892	17239	0.9536	0.9056	0.9846	8
ca-CondMat	108300	93439	0.9856	1.074	0.9023	5
com-youtube	1157828	2987624	1.8646	2.187	2.361	6
musae_git	37700	289003	1.0273	1.1236	1.075	5
roadNet-CA	1971281	2766607	3.132	2.8853	2.6363	263
Random	30K	10K	1.017	1.078	1.0293	8
Random	100K	10K	1.0533	1.0426	1.0126	4
Random	1M	1025	1.1997	1.277	1.1183	2
Random	1M	10K	1.1286	1.9726	1.496	2
Random	1M	100K	2.0693	1.9586	1.3246	3
Clique	100	4950	1.0526	0.9203	0.9983	2
Clique	1000	499500	0.947	0.8616	0.9166	2
Clique	10K	49995000	9.5026	9.4646	9.68	2
Alternate-Edge	100	50	1.115	0.976	1.098	2
Alternate-Edge	1000	500	0.9936	0.944	1.0303	2
Alternate-Edge	10K	5000	0.9913	0.868	0.9703	2
Alternate-Edge	100K	50K	1.0686	1.03	0.8613	2
Alternate-Edge	1M	500K	1.296	1.2453	1.3796	2
Alternate-Edge	10M	5M	5.4836	4.6967	4.8906	2
Alternate-Edge	50M	2.5M	21.4393	21.222	21.7033	2
Alternate-Edge	100M	50M	42.304	42.0686	41.9266	2
Alternate-Edge	500M	250M	💀	💀	💀	💀
Alternate-Edge	1B	500M	💀	💀	💀	💀
Chain	100	99	0.9973	1.0283	0.886	51
Chain	1000	999	1.0343	1.0726	1.1563	501
Chain	10K	9999	2.1616	2.0786	1.388	5001
Chain	100K	99999	33.447	31.3006	17.3906	49999
RMAT	1M	1M	1.443	1.6006	1.551	4

To preempt any critical readers from pointing out that we have the exact same figure repeated four times, and adding the comment that this is not the way to write research papers, let me clarify abundantly that the four figures are NOT the same, because each of them is happening at DIFFERENT instants of time, and this SHOULD be the way to write research papers, thank you very much.

Author: God, you betrayed us! Et tu, God!

God: Dude, don't expect me to spoonfeed you all the time. Just appeal to, um...

Author: Divine Intervention?

God: Yeah right!

Author: That's exactly what I did just now!

God (sheepishly): Ohhhh right! Ok well then, er... appeal to... um...

Author: Please don't tell Diabolical Intervention!

God: No no. Appeal to... you know, er..., yeah right, Curse of Higher Dimensions!

Author: What? That doesn't even make sense in the current context. Except maybe that we have been cursed by someone from Higher Dimensions. I hope you are getting the hint.

God: Ok ok fine. Look, the data is not all that bad. Some of them do support your hypothesis. Appeal to that. Appeal to Data Intervention!

Thus, thanks to Data Intervention, let us now analyze which inputs perform well with the Hybrid approach. We can make the following general observations from the data (any deviations in the data from these observations must be strictly considered to be noise in the data, which doesn't wish to bend itself to analysis or listen to reason):

1. For graphs where small number of nodes have high degree (like power-law graphs), the CC-time of the Push approach is the fastest although in terms of overall execution time, Hybrid sometimes does better.

2. For graphs with very small diameter, the Pull approach performs best. This is because the number of iterations is less, and hence walking through the neighbours (the phrase "walking through the neighbours" is not to be understood in a way that implies that we are ghosts — the neighbours are to be considered abstract entities, and "walking through" as an abstract operation of iteration) will not be too expensive. Hybrid approach also does fairly well, but the additional step of propagating minimum values to all neighbours does not gain much in terms of performance here because anyway, the values are going to converge in a very small number of iterations.

3. For graphs with large diameters, the Hybrid approach shines with the luminosity of a thousand suns! You are requested to wear protective sunglasses before looking at the results of its performance. Clearly, this performance boost is owing to the fact that the algorithm has to run through many iterations before reaching convergence, and hence the Hybrid approach would lead to savings in number of iterations in a large way, and per iteration, it helps in propagating the values further, which helps in the overall execution time as well.

4. The number of iterations with Hybrid approach is always less than (or equal to) the other two variants.

Having provided satisfactory explanations and observations, we can all finally go home happy. Ok, end of pap—

Reader: WAIT!

Author: Now what?

Reader: Um, those two inputs for which the program is crashing... Can't we do something about them?

Author: Well yes, hence the Grim Reaper has dragged away the dead cats from those rows. Because the corpses can't be allowed to remain there rotting throughout the paper.

Reader: No, I mean... The program is crashing or running out of time on those really large inputs.

Author: Look. The state that the program is in, it is basically a vegetable, and you can go ahead and chop it up and make gravy out of it and eat it whole.

DIGRESSION: THE BUTTERFLY EFFECT [11]

The flap of a butterfly's wings in one corner of the world might cause a hurricane in another corner of the world.

DIGRESSION-WITHIN-DIGRESSION:

Why can't you ask a hurricane to slow down?
Because if it did, it would no longer be a hurricane — it would be a relaxed-cane.

An innocuous pun made above has set the gears churning, leading to ideas for what needs to be done. The butterfly has flapped its wings and realizes the full enormity of what it has done, of being single-handedly (or rather, single-wingedly) responsible for creating a tornado on the opposite side of the globe, and frantically recalls Newton's Third Law and begs for Divine Intervention to invalidate the Law, or failing that, at least invent Newton's Fourth Law that provides an exemption clause to the Third Law.

DIGRESSION: Newton's Seventh Law of Motion

Newton's Fourth, Fifth and Sixth Laws of Motion are in constant Motion and cannot be pinned down or stated.

Because the word *chop* used above evokes a train of thought... What if we actually chop up the program? No, that probably doesn't make sense. But what if we chop up the graph?

5 RELATED WORK

[10]

END OF RELATED WORK

Author: Hey, you didn't really read that, did you?
Reader: No, I read it.
Author: Oh, come on!
Reader: Really! I swear, as sure as God is my witness, that I read it.
God: Ahem, I heard that.
Reader: G-G-God?
God: Yup, that's me. And you didn't read that paper.
Author: Aha!
Reader: G-G-God? I thought you didn't exist!
God: I heard that.
Devil: G-G-God? You exist?
God: I heard that too.
Reader: Are you vocabularily challenged?
God: I heard that.
Reader: Are you a machine preprogrammed to repeat just those words?

Reader: Whoa! What was that?
God: A display of my omnipotence.
Reader: But then, if you exist, how could you allow the Devil to exist as well?
God: Oh, um..., er..., **mumble* *mumble**... well, live and let live, you know?
Author: Ok, getting back on track, I urge you to read the reference.

[10]

Author: Now you have reached here too soon. You still didn't read it, did you? God, you can confirm this, can't you?
God: Look, I don't wish to be dragged into this.
Author: But God! You are the omnipresent and omniscient witness of everything! You know everything. You must have been a witness to whether the reader has read this or not.
God: Oh well, um..., er... you see, I wasn't.

Author: WHAT! BUT HOW? The Eye of God witnesses everything in the universe.

God: Er..., Um... Yeah that's true. But well, the Eye of God was a little preoccupied at the moment watching a really interesting sitcom on TV.

Reader: Ok, why don't you all trust me and believe me when I said that I did read the paper?

Author: Ok fine. Then let me ask you a test question to check if you really read the paper. Quick, tell me, what is Differential Privacy?

Reader: Um... Differential privacy is... er..., ok fine. I confess I didn't read it.

Author: Aha! Caught red-handed! Well, you really need to read that reference right now.

Reader: Ok ok fine. I will. But what is Differential Privacy anyway?

Author: I don't know. It certainly isn't mentioned in that reference, haha!

Reader:

6 UNRELATED WORK

While the reader is trying to come to grips, digesting the related work, here is a welcome digression to break the monotony. The reader is strongly encouraged to read the following works, because why should Jack have all work and no fun and become a dull boy, right?

[2], [3], [4], [5], [6], [7], [8], [9].

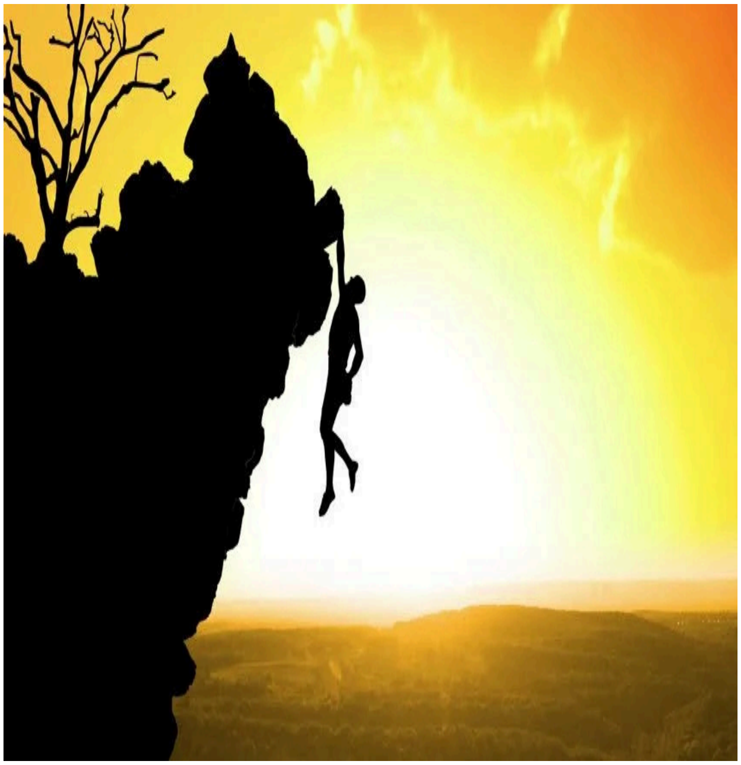
7 THE PARTING OF THE WAYS

Let us now return refreshed from that wondrous excursion, and come to the sudden unpleasant realization that the reader has probably forgotten the details of this paper. It is an uncomfortable realization on the part of the Author to know that the section on Unrelated Work was probably a mistake to be included, but by now, it is already too late and nothing can be done about it. Nevertheless, it has left the readers in a happy frame of mind, and since we know that the goal of life is happiness, we have nothing to complain about, I guess.

Meta-Moral of the Story

It is ok to make mistakes, maybe sometimes even in research papers.

However, to wrap things up, the Reader is strongly encouraged to ponder about partitioning the graph into multiple subgraphs. Because it is a well-established fact that Graphs are pervasive and growing in size by the day [1]. In other words, Why should all the terrorists be assigned to minions all at once? There probably aren't so many minions anyway, or else they would have been called *maxions*. So we can chop up the graph into different partitions. And let minions work on one such partition at a time. A natural question at this point is: How do we decide how to partition the graph? And how do each of the Push, Pull and Hybrid variants perform in the presence of partitioning?



8 CONCLUSION AND FUTURE WORK

Reader: Hey, what is that sound?
Author: What sound?
Reader: That ominous background music, that bodes something really bad is about to happen very soon. You are encouraged to scream to add to the special effects.
Author: Haha, you must be imagining things.
Reader: No, really. It's increasing in volume. And wait, do you hear that? That sounds like footsteps. A lot of footsteps. That's the sound of a stampede, footsteps of an unpacifiable mob, and they're approaching us!
Author: Yikes! You're right. But I haven't even had a chance to make my conclusion.
Reader: Let's run, because it's going to be a mob lynching.

Pant* *Pant So, to conclude, **pant** the connected components of a **pant** graph ***RUMBLE* *RUMBLE*** can be found **pant** ***RUMBLE* *RUMBLE*** no, NO NOOOO, I have more research directions to explore... I haven't even spoken about the Future Work, which is to explore Partitioning Strategies. Leave us alone! AAAAAAAAAAHHHHHHHHHH!

THE END

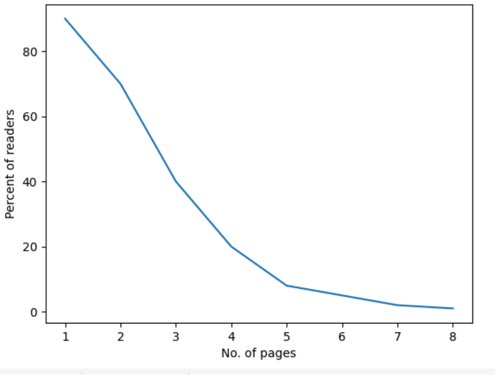
AUTHOR'S AFTERWORD
The Nameless Horrors of Research Papers and What You Can Do About Them (Answer: Nothing)

[12]

Yes yes, I know, I know. This is SO not the way to write a research paper, right? But apart from the fact that (a) it is informal, (b) there are way too many jokes, (c) digressions are the norm, (d) there are shocking twists and surprises often leading to catastrophic results, (e) there are life lessons, (f) there are dialogues, (g) there are comics strewn everywhere, (h) there are appearances of a Devil, and God as well, (i) it violates every possible known convention of any standard research paper..., there really is nothing wrong with this paper, and no reason why it should not be accepted as a research paper.

Research papers are known to be dry. The Sahara desert, if sentient, would feel deep jealousy towards most research papers in existence, and might even shed a few tears at its first position for dryness being usurped by many research papers, but then realize that in the process of shedding a few tears, it is no longer as dry as before, lowering its rank in this dry dryness list, and therefore, shed even more tears at this personal calamity, leading to an even further lowering of its rank, *ad infinitum.* THIS research paper, on the other hand, addresses this shortcoming by NOT being so dry. Well, I would rather have my paper be mosaic than prosaic, haha.

There are tons of research papers out there whose opacity is close to zero, inscrutable to all except for an enlightened few. The following graph illustrates:



The above graph presents an alarming picture. The coefficient of viscosity of research papers is so high that more than 50% of the readers have dropped off like flies by the end of page 3. And by the end of the paper, we only have 1 reader (which is probably why most papers have a page limit of 10-12). This is probably the Author who has written the paper himself/herself. This is alarming because it implies that not even the reviewers have been able to survive the vicious density of the paper to reach the end successfully. And often enough, it's either because (a) the paper is not self-contained, or (b) it oozes with terse jargon and deep math. Pure math is not everyone's cup of tea. I mean, it's like asking people to visualize an imaginary geometry of a three-headed dragon that curves in space-time and then proving that it cannot fly simply because breathing fire violates the law of commutativity. I can see you pure math enthusiasts moaning with pleasure at such abstractness, but this isn't for everyone. Because you see, here be the dragons.

Just getting through a page in any standard research paper of respectable quality requires immense concentration. Thus, we could say that the readers of such a paper are put in a concentration camp, haha! I hope to remedy this situation. The principle is simple: You do NOT want your readers to drop off dead like flies by the time they reach page 2 of your paper. Preferably you want them to die after they finish reading your paper. Whether as a result of the reading is of no consequence.

Reader: But this is a half-baked paper. It doesn't even go deep into the topic like any research paper should. Anybody who reads this will get half-baked knowledge.

Hey! This is supposed to be the Author's Afterword, and you aren't supposed to interrupt here. But oh well, seeing that the damage has been done anyway, let me reply to your point. Oh yes, this paper has half-baked knowledge, but that isn't the goal. This paper has half-baked knowledge because midway through the baking process, I suddenly realized that the knowledge wasn't meant to be baked, but rather fried in oil. So this resulted in an abrupt termination of the knowledge-baking process in order that it could be fried. So yes, this paper has half-baked knowledge, but that's ok because it also gives deep-fried knowledge. And it's been marinated as well. The crispiness is testimony to this. You can't have just one, haha!

Ok ok, fine... for a more formal treatment of the work presented in this paper, the interested and hardened reader is urged to refer to **[15]**.

My goal in writing this paper is straightforward: I wish research to be accessible to the motivated layman, even if it means that the actual quality of the research presented here is not dazzlingly novel. The goal is to etch the concepts presented here so deep into the reader that it becomes part of their DNA, so that the reader's offspring will be born with an innate understanding of these concepts. Ok, jokes aside, it takes one person to make a start to change long-established trends. Perhaps I have been such a person, and this paper has been perhaps an attempt in that direction, albeit maybe not the best attempt. This may not be a work of such monumental profundity and lucidity so as to catalyze a revolution in academia by dawn. But, at the very least... No animals have been harmed in writing this paper. However, a lot of rigour has been sacrificed. For the greater good. It is my hope that this will launch the "Make Research Papers Great Again" Movement. It is my hope that this will inspire many more readers to delve deeper into the field, contribute to it, and more importantly, dispense their learnings to others in an accessible way. May this not lose steam (because then, it would have to be called "Make Research Papers Great Again" Stagnant, haha). I have simply lit the torch. May the light of knowledge be passed on.

ACKNOWLEDGEMENTS

When one goes against the current to write a work of this sort, it is inherently obvious that nobody wishes to be seen associated dead or alive with the Author — imagine the ignominy! Anybody would conclude that a work of this sort must be the attempt of a demented and possessed researcher. But that would be wrong, because aren't all researchers possessed by default (by their institutions at the very least)? Perhaps the Author deserves to be cordoned off with a tape in the interests of public safety. So naturally the Author needs to be cautious while making acknowledgements to a work of this sort. Any names mentioned would automatically become the cast-out members of society, and those poor people, for no fault of theirs (except for making an appearance in an acknowledgement section) would be shunned by society, and probably die begging for foodgrains, clawing out their intestines in hunger because nobody is willing to have anything to do with them, much less provide them food. However, they are the brave few who have rendered help because they know what they are doing is right. So, for whatever it's worth, here goes:

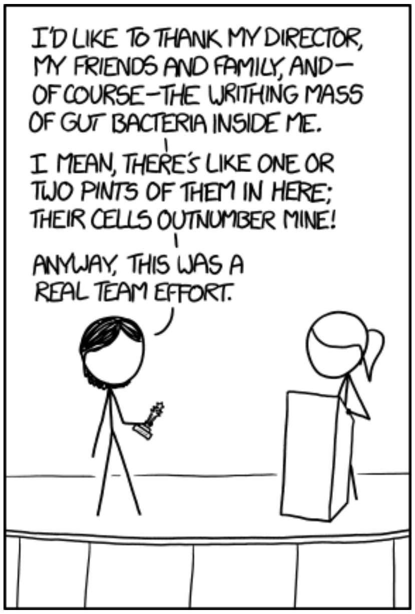
I owe the biggest debt of gratitude and acknowledgement to my mother, Geetha Chandrashekar. Whatever I am in life today is entirely owing to her and her family (special mention goes to her parents G S Chandrashekar and Shanthamma, her siblings, Sagar, Premalatha, and Raju, and their families (to take names, so as not to commit the sin of omission – Gayathri, Gouthami, Nikhil, Praveen, Jayashree, Vijay Chandra, Soujanya)). "The roots below the earth claim no rewards for making the branches fruitful" (Rabindranath Tagore) [9]. My mother is this kind of selfless person, who doesn't do a good turn for the sake of acknowledgement, but the branch in this case wishes to convey its gratitude to its root nevertheless.

A lot of teachers right from my childhood (from school to graduate college) have helped foster and encourage my creativity. To all of them, I prostrate with gratitude. I would especially like to acknowledge some of them: Govindarajan Sir, Rupesh Sir, Srikant Sir, Jayant Sir, Deepak Sir, Viraj Sir, NSK Sir, Arindam Sir, Rahul Sir, Dinesh Sir, Prema Ma'am, Jawahar Sir.

And more importantly, a lot of friends have been a staunch pillar of support for me through all my ups and downs. In particular, I would like to mention: Shubham Gupta, Chittaranjan, Varun, Sunil, Jeevan, Pradyumna, Hemanta, Gaurav, Srishty, Julian, AN, Nikitha, Shruti, Kavita, Pooja. There are others as well who wished not to be named. You know who you are, and I shall not embarrass you by naming you here. A huge shout-out to all of you, even if this does not reach your ears.

Now, I know I have ended up writing this Acknowledgement section in an unconventional way — certainly not in the way a research paper typically does. In other words, I have acknowledged not only those who were directly instrumental in bringing out this paper, like the way all research papers do. This is because I believe that any piece of monumental work is composed of many small parts, and it is unfair to only acknowledge the painters who put the

finishing touches of the monument, neglecting to mention all those other unsung contributors, without whom the monument would collapse. In other words...


I'D LIKE TO THANK MY DIRECTOR, MY FRIENDS AND FAMILY, AND— OF COURSE—THE WRITHING MASS OF GUT BACTERIA INSIDE ME.
I MEAN, THERE'S LIKE ONE OR TWO PINTS OF THEM IN HERE; THEIR CELLS OUTNUMBER MINE!
ANYWAY, THIS WAS A REAL TEAM EFFORT.
[13]

As mentioned in the beginning of this section, I have also intentionally left out the names of so many people (not because they have not contributed, and not because I have neglected to acknowledge them, but because I don't know how they would react to being publicly named in a work of this sort). If any of you wish to be named (and face the consequences thereof), if you think you have made a contribution and deserve acknowledgement, kindly let me know and I shall do so.

References

[1] Ullas 😄. "When Pull Comes To Shove... Do Both!" *2022*

[2] Rowling, Joanne K. "Harry Potter and the Philosopher's Stone." *Bloomsbury Publishing, 1997.*

[3] Rowling, Joanne K. "Harry Potter and the Chamber of Secrets." *Bloomsbury Publishing, 1998.*

[4] Rowling, Joanne K. "Harry Potter and the Prisoner of Azkaban." *Bloomsbury Publishing, 1999.*

[5] Rowling, Joanne K. "Harry Potter and the Goblet of Fire." *Bloomsbury Publishing, 2000.*

[6] Rowling, Joanne K. "Harry Potter and the Order of the Phoenix." *Bloomsbury Publishing, 2003.*

[7] Rowling, Joanne K. "Harry Potter and the Half-Blood Prince." *Bloomsbury Publishing, 2005.*

[8] Rowling, Joanne K. "Harry Potter and the Deathly Hallows." *Bloomsbury Publishing, 2007.*

[9] Tagore, Rabindranath. "Stray Birds." *1916.*

[10] Sabet, Amir Hossein Nodehi, Zhijia Zhao, and Rajiv Gupta. "Subway: Minimizing data transfer during out-of-GPU-memory graph processing." *Proceedings of the Fifteenth European Conference on Computer Systems, 2020.*

[11] https://en.wikipedia.org/wiki/Butterfly_effect

[12] https://www.smbc-comics.com/?id=3225

[13] https://xkcd.com/1543/

[14] SNAP Datasets: Stanford Large Network Dataset. https://snap.stanford.edu/

[15] At the time of this writing, this paper does not yet exist. The author apologizes for leading the reader through a dangling reference, but offers consolation by the promise that this reference will exist in the future (hopefully, subject to paper acceptance constraints).

exorcism(1)

NAME

exorcism - performs an exorcism (duh)

SYNOPSIS

exorcism [-l liturgy] [-d demon] [-S salt] [-t host] -ehsz

DESCRIPTION

Exorcism attempts to remove malevolent influences from your bits. Computers are infamously cursed. Despite being deterministic machines, they often display non-deterministic behavior. They sometimes require blood sacrifice to perform properly. Clearly these are signs of demonic possession.

Exorcism will perform a traditional exorcism, complete with ASCII bell, book, and candle.

-d, **--demon** *demon*

Which demon to attempt to cast out. See the Lesser Key of Solomon for a full list.

-e, **--detach-head** Allow HEAD to point to an unnamed neck.

-h, **--hex**

Output each removed curse as a hexadecimal string. Warning: Strings must be sanitized before piping to other programs to avoid curse transference. This can be accomplished via regular expressions, which are safe to use as they are already maximally cursed.

-l, **--liturgy** *liturgy*

Choose which incantation to use.

-S, **--salt** *salt*

Use a SHA-256 salt circle with the provided salt. If no salt is provided, a random 32-byte string will be used.

-s, **--silent** don’t emit an ascii bell character. A silent exorcism is not guaranteed to work.

-t, **--transmigrate** *host*

Instead of banishing the demon, migrate it into a new host. Host can be a fully qualified domain name or an IPv4 host (IPv6 is not supported and never will be).

–~Z~~zalgo-invoke/zalgo

EXIT STATUS

Returns 0 on success and 1 on failure. Or maybe it’s the other way around? We can never keep track.

BUGS

Only works on non-corporeal entities. Do not attempt to pronounce the list of no-parameter flags. SERIOUSLY.

SEE ALSO

daemon(3), hexdump(1), perl(1), ex(1), grep(1), printf(3), malloc(3)

JENSEN HUANG'S ABOUT TO MAKE YOU HIS CUSTOMER.

SEGA®

32ESQUE
IT'S ON-ISH

NVIDIA®

NVidia Sega 32esque presented by walkingsep, tokkot, ix exclusively for SIGBOVIK. Availability may vary in your market. ix@tchow.com.

SIGBOVIK

52 Analysis of A New Error Calculation Technique

Jevin Tong

Keywords: late, erroneous, hackneyed

53 A Third Thorough Investigation of the Degree to which the COVID-19 Pandemic has Enabled Subpar-Quality Papers to Make it into SIGBOVIK, by Reducing the Supply of Authors Willing to Invest the Necessary Effort to Produce High-Quality Papers

Shalin Shah

Keywords: SIGBOVIK, COVID-19, Lazy, Low-Effort

54 this is not a review

Huilian Sophie Qiu

Keywords: this, is, not, review

forgot to remove page numbers ach

Analysis of A New Error Calculation Technique

Jevin Tong

March 25th, 2022

1 Introduction

It is human to err. Unlike the cold steel of machinery, flesh and blood are prone to moments of weakness, such as forgetting about the existence of Sigbovik until the day that paper submissions are due. Furthermore, the unpredictability of humans makes it difficult to calculate the amount of errors that they can be expected to create in a given period of time. We[1] propose an experiment to ”eliminate a pair of avians while utilizing a singular rock”, as they say, by attempting to create a valid submission for Sigbovik in under 30 minutes using one average human test subject[1], in order to determine the average amount of grammatical and syntactical errors produced during the production of a paper.

2 Experiment Setup

To simulate a scenario of error-proneness, the subject was situated in a college apartment at 1:48 AM, and instructed to create a submission for Sigbovik 2022 after being reminded of the conference's existence a few mere minutes before being instructed to type the paper. Very little LaTeX experience was provided to the subject (although the internet as a resource was provided for the sake of LaTeX readability/reader benefit), and the subject's exposure to the formal stylings of academic writings has been relatively limited in scope. In order to compel the subject to type for the duration of the experiment, the subject was asked to think about the magnitude of their remaining homework load, thus motivating it to work on the paper in the spirit of ungodly procrastination.

3 Results

Due to the squiggly red lines, the subject was able to identify most of its spelling mistakes immediately and rectify those on the fly. As a result, most of the simple mistakes that subject would have made were obfuscated, which thus made the subject's biggest error its decision to choose error calculation as the subject of the paper. Fortunately, the subject was too sleep deprived to notice and correct any natural and egregious grammatical/syntactical[2] mistakes it had made that would not have been caught by the auto-speller. Furthermore, we observed that the subject made numerous revisions to the paper regarding the premise of the paper, as during brief moments of lucidity, it realized the futility of the paper's initial premise and frantically attempted to steer the paper in the direction of a better joke. From the paper produced, we can tell that about multiple mistakes were produced over the course of 30 minutes, suggesting that humans are pretty unpredictable, I guess.

4 Further Research

Due to the subject's immense propensity for procrastination, further studies into the error production of the subject will be easy to reproduce.

[1] I, me, myself, etc.

[2] The difference between the two being unknown to the writer

A Third Thorough Investigation of the Degree to which the COVID-19 Pandemic has Enabled Subpar-Quality Papers to Make it into SIGBOVIK, by Reducing the Supply of Authors Willing to Invest the Necessary Effort to Produce High-Quality Papers

Shalin Shah

Carnegie Mellon University

April 1, 2022

Abstract:

Based on this paper's inclusion in the proceedings of SIGBOVIK 2022 (despite it being barely modified from our similarly lazy yet accepted submissions to SIGBOVIK in 2020 and 2021), we find that the COVID-19 pandemic has in fact enabled subpar-quality papers to make their way into the proceedings of SIGBOVIK, even more so than in previous years, presumably by reducing the supply of authors willing to invest the necessary effort to produce high-quality papers.

Introduction:

Y'all know what COVID-19 is.

Methods and Materials:

You're looking at the materials. Note that, in order to emphasize the subpar quality of this paper, we have opted to use extremely lazy Microsoft-Word default formatting, rather than LaTeX. Also, we have restricted the contents of this paper to a single page, to highlight its lack of substance. Meanwhile, our method was to simply submit this paper to SIGBOVIK 2022 and see what happened.

Note that this paper is in fact almost exactly the same as what we submitted to SIGBOVIK in 2020 and 2021, with only minor updates. With each passing year, our submission of this paper to SIGBOVIK becomes an even lazier, lower-effort endeavor, and embodies an even greater lack of creative originality. Thus, this paper's acceptance into SIGBOVIK 2022 convincingly demonstrates that SIGBOVIK's standards have fallen even lower since last year.

Results:

As evidenced by the fact that you're currently reading this in the SIGBOVIK 2022 proceedings, this paper successfully made it into the SIGBOVIK 2022 proceedings.

Discussion:

The results indicate that SIGBOVIK's standards of quality have indeed decreased steadily since 2019, presumably due to the COVID-19 pandemic decreasing the supply of authors willing to invest the necessary effort to produce high-quality paper submissions.

Conclusions:

In conclusion, COVID-19 sucks.

References:

n/a

Decision: Strong reject

it is 20:34 set on apr 1st no im not kidding i just started writing this review though i have pondered on it for a while i think i should say something to make myself seem capable of reviewing this masterpiece but i have to admit that usually reviewers dont have the ability to make them sound smart to avoid teaching you something that even i myself dont really understand i decide to shut up now

wait was there a paper for me to review

oh never mind the paper the paper is never part of a review im a mature reviewer now ive learned to reject all papers i review
sorry i wrote this in a hurry and probably forgot some punctuation
do i need a punchline