the association for computational heresy

presents

a record of the proceedings of

SIGBOVIK 2019

the thirteenth 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 chris yu
cover art by chris yu
cover art by chris yu
cover art by chris yu

carnegie mellon university

pittsburgh, pa

april 1, 2019

Association for Computational Heresy

Advancing computing as Tomfoolery & Distraction

SIGBOVIK

A Record of the Proceedings of SIGBOVIK 2019

ISSN 2155-0166

April 1, 2019

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

SIGBOVIK 2019

Message from the Organizing Committee

Greetings friends, colleagues, and complete strangers. It is with great pride that we welcome you to the 0xDth annual Special Interest Group on Harry Q. Bovik, held in celebration of Harry Q Bovik's $10^{somenumberbiggerthan2}$th birthday. Normally this is where we, the "committee", would wax poetic about features added[1], records broken, and facts fun. However, this year, we have decided to take the highly efficient and not at all lazy route of delegating the task of writing this message to committee members of the past. A sort of temporal passing of the buck, one might say. Thus I present to you, in reverse chronological order, some messages.

- "Best conference I have chaired so far, also the question mark is implicit at the end of all the survey questionspunctuationisfortheweak" — *General Chair 2019*
- "much as even mentioned in the paper; well, that will teach me to forget punctuation at the end of my survey questions, won't it?" — *General Chair 2018*
- "SIGBOVIK, for those with a special interest in groups-BOVIK. Like many special interests, I consider it clinically significant." — *General Chair 2017*
- "I wish all conferences were as ~~easy to get into~~ prestigious as SIGBOVIK" — *General Chair 2014*
- " 'i'm sorry i derelicted my duties to promote sigbovik on twitter this year but i see you have 30ish submissions already even without me so HRMPH WHO NEEDS ME um i mean well done!' " — *General Chair 2012 (note: 40ish actually, end note)*
- "SIGBOVIK is exactly what I had in mind. —Edgar Allen Poe" — *General Chair 2008*
- "I like to bovik bovik" — *General Chair 2007*

And with that, the papers.

[1] Despite my noble intention to dispatch with the labored, formulaic frontmatter of yestersyear, I was thwarted by a proceedings chair who insisted that I add an explanation of the unwittily-named "unwitting participation ribbon" (🎗), an unwelcome brand we've affixed to each paper determined after careful scrutiny to have included a genuine artifact, thereby furthering the admirable causes of open science and fruitful procrastination.

Academic Games

: Gamification **3**

: Let's get this party started! **11**

: Survival advice from a computer scientist **33**

: Security and privacy **53**

: Machine learning **59**

: Architecture: Faster and hotter **85**

: Chess **119**

: Simple in theory 131

: Languages 137

: Error-correcting codes 153

: Measurement studies 169

: Back to the future 187

: Pop culture 203

Gamification

Aumann Agreement by Combat

Travis Hance

Abstract

The celebrated Aumann's Agreement Theorem [2] shows that two rational agents with the same priors on an event who make different observations will always converge on the same posteriors after some civilized conversation over tea. Furthermore, they will come to agree even if they do nothing other than simply state their posteriors over and over again.

However, this protocol is widely criticized for being too boring. We therefore introduce a more exciting alternative, which we name *Aumann Agreement by Combat*.

1 Introduction

Informally, Aumann's Agreement Theorem [2] states that two honest Bayesian agents can come to agreement by stating their priors over and over again. (Here, the term *agent* simply means "a person who observes things and has opinions." However, it is more exciting to imagine them as *secret agents*, so we will do so for this paper.)

To take example from Aumann's paper, suppose the agents are investigating the fairness of a particular coin (say, a very rare coin involved in a museum heist). We will call the agents A and B, but of course they are secret agents, so what names these letters stand for is a mystery. The coin is parameterized by a value p, the probability that it will land heads when it is flipped. Each agent begins with the same prior, in this example the uniform prior of p over the interval $[0, 1]$, because the uniform prior is a reasonable prior to assume about a coin that one knows nothing about.

If A flips the coins and sees heads, then her Bayesian calculation will conclude that the probability of the *next* flip being heads will be $2/3$. If B flips and sees tails, they he will think the probability is $1/3$. They are now in disagreement. However, if these agents then meet in a tavern to discuss, then A needs merely to state that her probability is $2/3$, and B will conclude that she must have seen heads in his single flip. With all of that information, she now concludes that the probability of another flip being heads will be $1/2$. Observers will assume that these numbers are all part of a secret code, but A will understand that B has computed the true probability, and she will update her probability to $1/2$ as well. Both agents are now in agreement.

Of course, this scenario assumed that both agents knew that the other would be flipping the coin only once. In more complicated scenarios, the agents may have variable methods of observation, for example, they might flip multiple times, in which case their observations should be weighted more, or they might not. Still, as long as they have the same priors on said methodology, they will be able to come to agreement through a sequence of statements.

Unfortunately, this methodology of merely stating opinions is boring. Furthermore, everybody knows that the goal of arguing is not to reach the truth, but rather to win. Therefore, we have developed a new protocol, *Aumann Agreement by Combat* that alleviates these problems.

2 Aumann Agreement by Combat

Our new protocol works as follows.

1. Agents A and B start with the same priors.
2. A and B each go out and make observations.
3. A and B meet to discuss the probability of some event E. Each agent has some initial estimate of the probability of E.
4. Players alternate in turns, continuing until their estimates match. On each turn, an agent has a choice to either:
   - State their current estimate of E. The other player must use Bayesian reasoning to update their own estimate.
   - Declare COMBAT. The agents will fight it out, and at the end, the loser will adopt the winner's estimate as their own.

The COMBAT phase has no rules. Any player is free to try to win by any means necessary, and they are encouraged to employ all of their wit and resources to the fullest extent. In the base rules, the battle ends when one player loses consciousness, although the players may also agree to a different ending condition, such as getting a flag to one's base, death, or the outcome of a poetry competition.

With this definition in place, we come to our main theorem.

Theorem 1. *After an execution of the Aumann Agreement by Combat protocol, both agents will agree on the correct probability of the event E.*

Proof. Might makes right. □

3 Complexity Analysis

In [1], Aaronson analyzes the complexity of Aumann Agreement, showing that $O(1/\varepsilon^2)$ bits of communication are sufficient to agree within ε.

An advantage of our new protocol is that the analysis is much simpler. The COMBAT phase might take an arbitrary amount of time and resources, depending on the strategies that the agents employ. They might stake the COMBAT phase on the outcome of an intergalactic war, for example. Therefore, there is no complexity bound on time, space, communication, energy, or anything else.

4 Conclusion

We expect that many scholars will welcome this formalization of the game that they already try to play.

References

[1] S. Aaronson. The complexity of agreement. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 634–643, New York, NY, USA, 2005. ACM.

[2] R. J. Aumann. Agreeing to disagree. *Ann. Statist.*, 4(6):1236–1239, 11 1976.

Monetization of Development Tools for Fun and Profit

Albin Eldstål-Damlin
Your Company Here
Reasonable rates apply
albin@legit.name

***Abstract*—In this paper we propose methods and strategies to raise profit from freely available and open-source development toolchains such as GCC. We illustrate techniques to maximize ~~player~~ developer engagement and drive further purchases once the system is in place.**

***Index Terms*—monetization, microtransactions, free-to-play, gamification**

I. Introduction

In the past decade, monetization of free content has become a leading business strategy for companies in a range of tech industries such as computer games and mobile apps. A wide variety of techniques have been developed and refined to drive users to continuously pay for additional content, either in a storefront fashion or using a more randomized ”loot-box” approach. This has caused a surge of ”Freemium” titles, requiring no up-front purchase and recouperating development costs by selling in-game items.

A previously untapped market segment is that of *software development tools*, such as compilers, static analysis tools, debuggers, etc. Many of these are available either free of charge or under an open-source license, making them perfect candidates for freemium-style monetization.

In this paper, we examine g++, the C++ compiler of the GNU Compiler Collection (GCC). Consulting its manual [1], we find that the standard distribution comes with an ample selection of options and switches; this suggests that there is room for expansion.

We explore the possibilities of adding microtransactions to the g++ frontend.

II. User Motivation

The first problem is convincing the user to enter the ecosystem, taking the step from Libre to Freemium. To entice, we must introduce a *killer feature* that is not available elsewhere. What sets a *killer* feature apart from a regular feature is that the killer feature is unique and indispensable. This feature can be entirely cosmetic (such as a novel output decoration), but it is preferable to give the user some qualitative improvement. To come up with such a feature, we identify a common problem users have with our chosen product and devise a solution.

In the case of g++, a common complaint is the verbosity and obscurity of some error outputs. In many small- and mid-sized codebases, a typo can lead to error messages exceeding the size of the code! Furthermore, it isn't always clear to the novice programmer what the cause is of these pages of output. Listing 1 shows an example of an STL type error [2], with the full error message from g++ in appendix A.

```
#include <vector>
#include <algorithm>

using std::vector;
using std::find;

int main()
{
  int a;
  vector< vector <int> > v;
  vector< vector <int> >::const_iterator it
      = std::find( v.begin(), v.end(), a );
}
```

Listing 1. A program with confusing errors

A good candidate for a killer feature, then, is error output of improved quality, readability and accuracy. The implementation of such a feature is beyond the scope of this article.

III. Intermission

If you have a software project, monetization effort or just want to show off pictures of your cats to the world, you **need** a website. Don't have any coding skills? Don't know where to start? Check out SquarePeg, one of the world's leading providers of all-in-one web hosting solutions. SquarePeg lets you effortlessly create your site using one of *four* beautiful templates. It's as easy as drag-and-drop. Almost nothing to install, almost nothing to update, almost ever! Visit http://www.squarepeg.com/sigbovik for 20% off your first purchase today!

IV. In-tool currency

A key technique to drive purchases is an intermediate single-purpose currency, which can either be earned by using the tool or directly purchased for real-life money. Such a currency serves to disconnect the actual cost of offered add-ons from the apparent cost, in addition to locking in a greater amount of real-world money by inconvenient exchange rates.

In our examples, we introduce the g++-specific ”L2 Cash” (L$) and present all purchase options to the user (except for L$ itself) in L$.

The user is rewarded in small amounts of L$ for various normal use of the software (for example 1L$ per 10 seconds spent compiling code) as well as extraordinary achievements (compile with substantial changes without errors on the first try? 1L$ per 10 lines of code!).

Another avenue for monetization is to sell ad-space and reward the user for being exposed to advertising. Targeting g++ opens the non-traditional option of modifying the standard library, for example by randomly including advertising in printf() output in exchange for L$.

V. Suitable Features

Traditionally, premium add-ons can be divided into three categories along a spectrum: *Cosmetic*, *Convenience* and *Pay-to-Win*. The special nature of a compiler also offers a fourth option which is useful to us: *Non-standard language features*.

Cosmetic features do not alter the user's experience, only the apperance of some elements. In an online game, this may be a special player avatar or some rare piece of equipment that nevertheless does not afford the player any advantage. There are relatively few possibilities for a purely cosmetic add-on to a productivity tool such as a compiler.

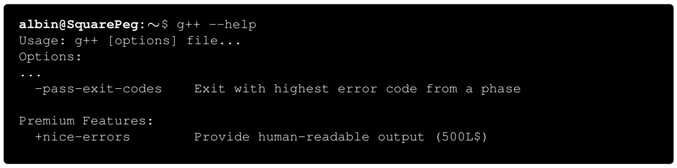
Convenience features can automate or simplify otherwise tedious tasks, saving the user time but not otherwise providing any competitive edge. An example might be automation of *grinding*, mindless mass-production work. In a development setting, this is akin to well-crafted preprocessor macros and automatic generation of boilerplate code.

Pay-to-Win features directly improve a user's chance of success. In a competitive game, this may be a stronger weapon or a higher-capacity backpack. These features are generally ill-received, as they are perceived to throw off the balance of the game. Since development tools are usually single-player, or cooperative multiplayer, these features are better thought of as productivity-enhancements.

Non-standard language features offer an avenue for lock-in, by luring the user into writing programs that will not work without our premium add-ons. This is ideal for monetization, and a particularly good feature could even warrant recurring payment. We will also exploit features like this for licensing reasons, detailed in Section VII.

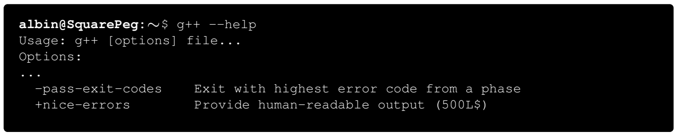
VI. User Interface

The most obvious way to expose new extensions to the user is via command-line switches. We use the + prefix to show features that can be purchased, together with their price.



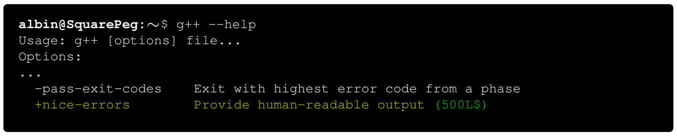
```
albin@SquarePeg:~$ g++ --help
Usage: g++ [options] file...
Options:
...
  -pass-exit-codes    Exit with highest error code from a phase

Premium Features:
  +nice-errors        Provide human-readable output (500L$)
```

A common optimization is to sneak the premium options in among the free ones, to make them more difficult to ignore.



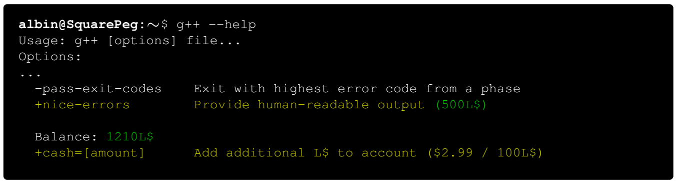
```
albin@SquarePeg:~$ g++ --help
Usage: g++ [options] file...
Options:
...
  -pass-exit-codes    Exit with highest error code from a phase
  +nice-errors        Provide human-readable output (500L$)
```

To drive conversion rate, however, it is useful to advertise the available options more strongly. A color hint is a good start.



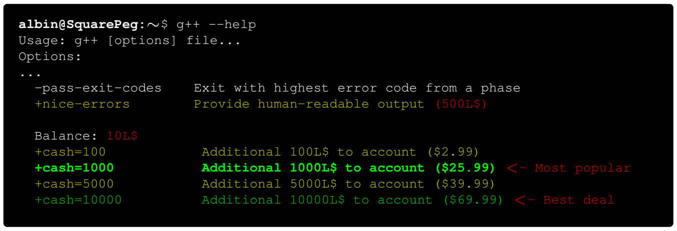
```
albin@SquarePeg:~$ g++ --help
Usage: g++ [options] file...
Options:
...
  -pass-exit-codes    Exit with highest error code from a phase
  +nice-errors        Provide human-readable output (500L$)
```

At the end of every output, we append the user's current account balance. This provides a reminder that the player is earning while they play. Of course, we also remind them how to easily increase their account balance.



```
albin@SquarePeg:~$ g++ --help
Usage: g++ [options] file...
Options:
...
  -pass-exit-codes    Exit with highest error code from a phase
  +nice-errors        Provide human-readable output (500L$)

  Balance: 1210L$
  +cash=[amount]      Add additional L$ to account ($2.99 / 100L$)
```

Another well-known strategy is to provide tiered pricing, i.e. a better exchange rate for bulk currency purchase. By identifying beginners (more likely to make small purchase) and power-users (more likely to make bulk purchases), we can gently nudge them toward a transaction.



```
albin@SquarePeg:~$ g++ --help
Usage: g++ [options] file...
Options:
...
  -pass-exit-codes    Exit with highest error code from a phase
  +nice-errors        Provide human-readable output (500L$)

  Balance: 10L$
  +cash=100            Additional 100L$ to account ($2.99)
  +cash=1000           Additional 1000L$ to account ($25.99) <- Most popular
  +cash=5000           Additional 5000L$ to account ($39.99)
  +cash=10000          Additional 10000L$ to account ($69.99) <- Best deal
```

VII. Skirting Open-Source Licensing

g++ is released under the GNU General Public License (GPL), which requires that any modification or addition is also released under the same license. To counteract this, we devise a strategy for compliance with *the letter* of the license while side-stepping the *spirit*.

By ensuring that our premium features are implemented with heavy reliance on our own non-standard language features, we prevent the spread of our proprietary modules to non-paying users. By extension, if we also hide the documentation of these non-standard features behind our pay-wall, we inhibit non-paying users trying to port the released source code to free tools.

References

[1] G. Project, "Gcc manual: Invoking gcc." https://gcc.gnu.org/onlinedocs/gcc-8.1.0/gcc/Invoking-GCC.html#Invoking-GCC, 2018.

[2] Kebs, "Code golf entry." https://codegolf.stackexchange.com/a/10470, 2016.

APPENDIX A
TYPICAL G++ OUTPUT

CONFIDENTIAL COMMITTEE MATERIALS

SIGBOVIK 2019 Paper Review

Paper 1: Monetization of Development Tools for Fun and Profit

ReviewIt Free
Rating: [Wait remaining: 2h35m; Reveal Now: 25R$]
Confidence: [Upgrade Your Review Experience With Confidence Levels: 75R$]

This paper has been reviewed using ReviewIt Free, a service of ReviewIt enterprises. ReviewIt Free provides the same great review feedback to authors as ReviewIt Enterprise, ReviewIt Premium, ReviewIt Pro, ReviewIt Gold, and ReviewIt Solid Gold with a few limitations:

- first, all reviews begin with this block of text and a short but mandatory advertisement;
- second, authors that wish to immediately view a numerical summary score of their review can do so for a small micro-payment – the score will otherwise be revealed several hours after the review release, but before the committee needs it to make accept/reject decisions;
- third, the reviewer confidence rating is also available for a larger micropayment.

These limitations ensure that ReviewIt enterprises can continue to provide great reviews to all our customers. We have carefully considered all of our micropayments and decided that any diffuse negative influence they may have on the progress of science is certainly more than made up for by the concentrated positive influence they have on the balance in our wallets.

Ad-Blocker Detected! No review text displayed. Please disable your ad-blocker so that we can force you to watch a five minute edutisement from PraegerU; monthly subscribers don't need to watch these advertisements.

Hey authors, not happy with your review results? Upgrade your reviewing experience with these extra features:

- Proper grammar, spelling, and complete sentences. [100R$]
- No statements of opinion couched as fact. [200R$]
- Sensible feedback that considers that paper authors are people, too, and are generally making a good-faith effort to move science forward. [currently unavailable]
- Specific, constructive feedback. [10000R$]
- Re-roll the review hoping for better drops. [submit the paper next year]

Let's get this party started!

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess

Elo World, a framework for benchmarking weak chess engines

DR. TOM MURPHY VII PH.D.

CCS Concepts: • **Evaluation methodologies → Tournaments**; • **Chess → Being bad at it**;

Additional Key Words and Phrases: pawn, horse, bishop, castle, queen, king

ACH Reference Format:
Dr. Tom Murphy VII Ph.D.. 2019. Elo World, a framework for benchmarking weak chess engines. 1, 1 (March 2019), 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Fiddly bits aside, it is a solved problem to maintain a numeric skill rating of players for some game (for example chess, but also sports, e-sports, probably also z-sports if that's a thing). Though it has some competition (suggesting the need for a meta-rating system to compare them), the Elo Rating System [2] is a simple and effective way to do it. This paper is concerned with Elo in chess, its original purpose.

The gist of this system is to track a single score for each player. The scores are defined such that the expected outcomes of games can be computed from the scores (for example, a player with a rating of 2400 should win 9/10 of her games against a player with a rating of 2000). If the true outcome (of e.g. a tournament) doesn't match the expected outcome, then both player's scores are adjusted towards values that would have produced the expected result. Over time, scores thus become a more accurate reflection of players' skill, while also allowing for players to change skill level. This system is carefully described elsewhere, so we can just leave it at that.

The players need not be human, and in fact this can facilitate running many games and thereby getting arbitrarily accurate ratings.

The problem this paper addresses is that basically all chess tournaments (whether with humans or computers or both) are between players who know how to play chess, are interested in winning their games, and have some reasonable level of skill. This makes it hard to give a rating to weak players: They just lose every single game and so tend towards a rating of $-\infty$.[1] Even if other comparatively weak players existed to participate in the tournament and occasionally lose to the player under study, it may still be difficult to understand how this cohort performs in any absolute sense. (In contrast we have "the highest ever human rating was 2882," and "there are 5,323 players with ratings in 2200–2299" and "Players whose ratings drop below 1000 are listed on the next list as 'delisted'." [1]) It may also be the case that all weak players always lose to all strong players, making it unclear just how wide the performance gap between the two sets is. The goal of this paper is to expand the dynamic range of chess ratings to span all the way from extremely weak players to extremely strong ones, while providing canonical reference points along the way.

Appears in SIGBOVIK 19119 with the title inflation of the Association for Computational Heresy; *IEEEEEE!* press, Verlag-Verlag volume no. 0x40-2A. 1600 rating points.
Author's address: Dr. Tom Murphy VII Ph.D., tom7@tom7.org.

[1] Some organizations don't even let ratings fall beneath a certain level, for example, the lowest possible USCF rating is 100.

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess

2 ELO WORLD

Elo World is a tournament with dozens of computer players. The computer players include some traditional chess engines, but also many algorithms chosen for their simplicity, as well as some designed to be competitively bad.

The fun part is the various algorithms, so let's get into that. First, some ground rules and guidelines:

- No resigning (forfeiting) or claiming a draw. The player will only be asked to make a move when there exists one, and it must choose a move in finite time. (In practice, most of the players are extremely fast, with the slowest ones using about one second of CPU time per move.)
- The player can retain state for the game, and executes moves sequentially (for either the white or black pieces), but cannot have state meaningfully span games. For example, it is not permitted to do man-in-the-middle attacks [4] or learn opponent's moves from previous rounds, or to get better at chess. The majority of players are actually completely stateless, just a function of type `position` ⟶ `move`.
- The player should try to "behave the same" when playing with the white or black pieces. Of course this can't be literally true, and in fact some algorithms can't be symmetric, so it's, like, a suggestion.
- Avoid game-tree search. Minimax, alpha–beta, etc. are the correct way to play chess programmatically. They are well-studied (i.e., boring) and effective, and so not well suited to our problem. A less obvious issue is that they are endlessly parameterizable, for example the search ply; this leaves us with a million things to fiddle with. In any case, several traditional chess engines are included for comparison.

det	The player is deterministic
trad	Traditional approach to chess (e.g. engine)
🌱	Vegetarian or vegetarian option available
canon	A canonical algorithm!
state	Stateful (not including pseudorandom pool)
asym	Asymmetric

Fig. 1. Key

2.1 Simple players

`random_move`. We must begin with the most canonical of all strategies: Choosing a legal move uniformly at random. This is a lovely choice for Elo World, for several reasons: It is simple to describe. It is clearly canonical, in that anyone undertaking a similar project would come up with the same thing. It is capable of producing any sequence of moves, and thus completely spans the gamut from the worst possible player to the best. If we run the tournament long enough, it will eventually at least draw games even against a hypothetical perfect opponent, a sort of Boltzmann Brilliancy. Note that this strategy actually does keep state (the pseudorandom pool), despite the admonition above. We can see this as not really state but a simulation of an external source of "true" randomness. Most other players fall back on making random moves to break ties or when their primary strategy does not apply. canon

`same_color`. When playing as white, put pieces on white squares. Vice versa for black. This is accomplished by counting the number of white pieces on white squares *after* each possible move, and then playing one of the moves that maximizes this number. Ties are broken randomly. Like many algorithms described this way, it tends to reach a plateau where the metric cannot be increased in a single move, and then plays randomly along this local maximum (Figure 2). This particular strategy moves one knight at most once (because they always change color when moving) unless forced; on the other hand both bishops can be safely moved anywhere when the metric is maxed out.

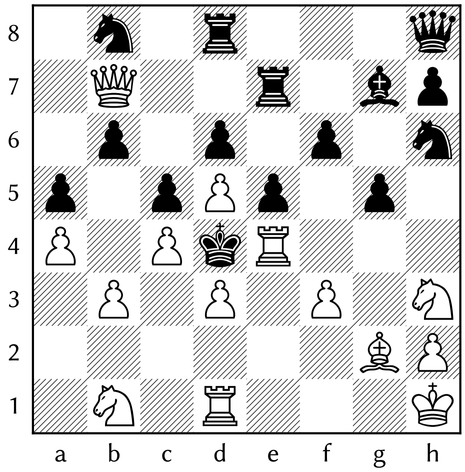


Fig. 2. `same_color` (playing as white) checkmates `same_color` (playing as black) on move 73. Note that since white opened with Nh3, the h2 pawn is stuck on a black square. Moving the knight out of the way would require that move to be forced.

`opposite_color`. Same idea, opposite parity.

`pacifist`. Avoid moves that mate the opponent, and failing that, avoid moves that check, and failing that, avoid moves that capture pieces, and failing that, capture lower value pieces. Break ties randomly. This is one of the worst strategies, drawing against players that are not ambitious about normal chess pursuits, and easily losing to simple strategies. On the other hand, it does rarely get forced into mating its opponent by aggressive but weak players.

`first_move`. Make the lexicographically first legal move. The moves are ordered as ⟨`src_row`, `src_col`, `dst_row`, `dst_col`, `promote_to`⟩ for white (rank 1 is row 0) and the rows are reversed for black to make the strategy symmetric. Tends to produce draws (by repetition), because knights and rooks can often move back and forth on the first few files. det

`alphabetical`. Make the alphabetically first move, using standard PGN short notation (e.g. "a3" < "O-O" < "Qxg7"). White and black both try to move towards A1. asym det

`huddle`. As white, make moves that minimize the total distance between white pieces and the white king. Distance is the Chebyshev distance, which is the number of moves a king takes to move between squares. This forms a defensive wall around the king (Figure 3).

`swarm`. Like `huddle`, but instead move pieces such that they are close to opponent's king. This is essentially an all-out attack with no planning, and manages to be one of the better-performing "simple" strategies. From the bloodbaths it creates, it even manages a few victories against strong opponents (Figure 4).

`generous`. Move so as to maximize the number of opponent moves that capture our pieces, weighting by the value of the offered piece (♙ = 1, ♗ = ♘ = 3, ♖ = 5, ♕ = 9). A piece that can be catpured multiple ways is counted multiple times.

`no_i_insist`. Like `generous`, but be overwhelmingly polite by trying to *force* the opponent to accept the gift of material. There are three tiers: Avoid at

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess

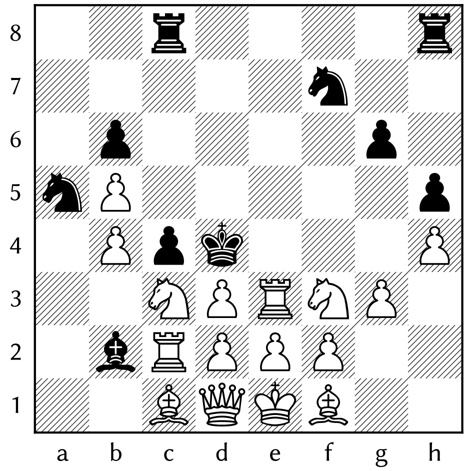


Fig. 3. huddle (white) checkmates pacifist on move 158 with substantial help. Note that white's distal pawns have advanced; these are actually the same distance from the King as they would be in their home row, since the distance metric includes diagonal moves.

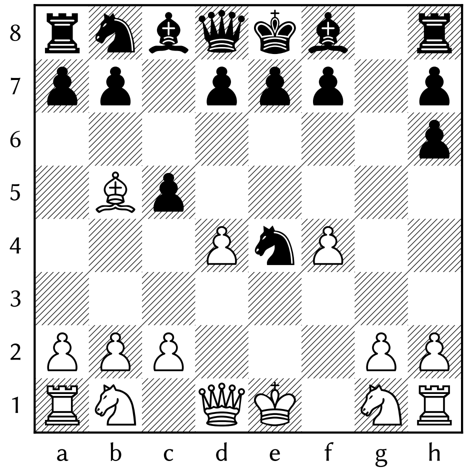


Fig. 4. swarm (white) vs. stockfish1m_r4096 with black to move after 1. d4 Nf6 2. Bh6 gxh6 3. f4 c5 4. e4 Nxe4 5. Bb5. Black blunders 5…f6??, which must have been a random move as swarm immediately wins with 6. Qh5++.

all costs mating the opponent (and moreso checkmating). Stalemate is polite, but it is more canonical for two polite players to form a draw by repetition, from continually offering up material to one another and declining it. Next, avoid situations where the opponent can refuse our gift; among these, prefer the move where the opponent must capture the highest value piece. Finally, prefer moves where the expected value of the offered material (i.e. against random play) is highest. (This means that if there are three moves, and one captures a rook but the others capture nothing, the value is 5/3.) This strategy almost never wins, but is not among the worst players, since it often forces a draw by exchanging all its pieces.

reverse_starting. This player thinks that the board is upside-down, and as white, tries to put its pieces where black pieces start. Since here we have a specific configuration in mind, we can produce a distance metric by computing the total distance from each piece to its desired destination. Each piece uses a different distance metric; Chebyshev distance for the King, Manhattan distance for the Rook, and a pretty weird function for the Knight [3]. Trying to move the pieces into the reversed starting position often causes some conflict since the black pieces are already there, but can also end peacefully (Figure 5).

cccp. Prioritize moves that Checkmate, Check, Capture or Push, in that order. Push means to move pieces as deep as possible into enemy territory (without any regard for their safety). Ties are broken deterministically by the source/destination squares, so this one is technically asymmetric. (det) (asym) (За здоровье!)

suicide_king. Take a random move that minimizes the distance between the two kings. Putting

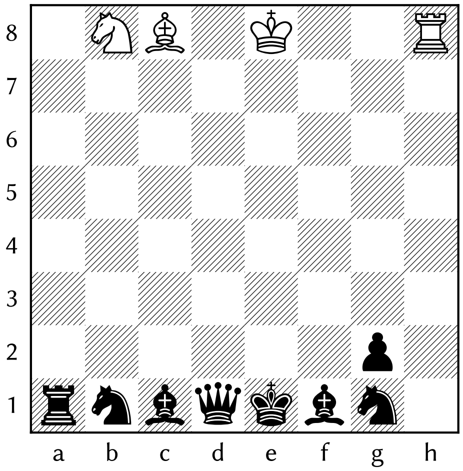


Fig. 5. `reverse_starting` draws against itself by repetition, both players having happily moved their surviving pieces into the correct spots.

one's king out in the open is very unprincipled (Figure 6), but it does produce enough pressure to win against unambitious opponents.

`sym_mirror_y`. As white, try to maximize symmetry when flipping vertically. Zero penalty for opposing e.g. a ♚ with a ♔, but a small penalty when the pieces are not the same type, and a larger penalty when they are not opposite colors. The starting position is already symmetric this way, so this usually has the effect of copying the opponent's moves when possible. As "copy your opponent's moves" is a common (but underspecified) strategy discovered by many children, this player is close to being canonical. However, it admits a bit too much arbitrary choice in the penalties assigned.

`sym_mirror_x`. As `sym_mirror_y`, but maximize symmetry when flipping horizontally. This does not make much chess sense, but can produce aesthetic arrangements.

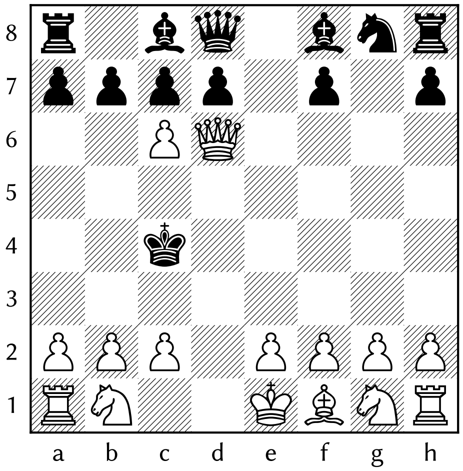


Fig. 6. `suicide_king` (black) dramatically failing against `cccp`'s slightly more principled play, after 1. d4 g5 2. Bxg5 Nc6 3. Bxe7 Kxe7 4. d5 Kd6 5. dxc6+ Kc5 6. Qd6+ Kc4. White delivers a discovered mate with 7. e4++.

`sym_180`. As `sym_mirror_y`, but maximize symmetry under 180° rotation of the board (Figure 7). An emergent priority is to "castle" with the king and queen to "fix" them.

`min_oppt_moves`. Take a move that minimizes the number of resulting legal moves for the opponent, breaking ties randomly. This is a beautifully simple approach that generalizes many chess principles: Occupying space reduces the destination squares available to the opponent; capturing their pieces reduces the number of their pieces that they can move; pinning pieces or checking the king further reduces the legal moves; and mating the opponent is the best possible move.[2] Among the players in the paper, this one is Pareto efficient in terms of its simplicity and effectiveness. canon

[2] However note that it does not distinguish checkmate and stalemate, despite these having very different results.

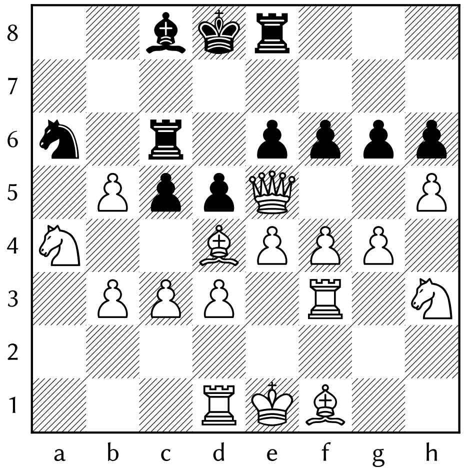


Fig. 7. `sym_180` (white) vs. `pacifist` after 66 moves. Note that the position is not quite rotationally symmetric, but is close given the material.

`equalizer`. Prefer to move a piece that has been moved the fewest times, and then prefer moving to a square that has been visited the fewest times. Castling counts as moving both the king and rook, and visiting both destination squares. This is the first strategy described that keeps meaningful state. (canon) (state)

2.2 Fate-based players

If we keep state, then we can track the location of each piece as it moves around the board (allowing us to distinguish the two rooks, or follow a pawn as it promotes). We can then use statistics on the average fates of each piece over hundreds of millions of games to guide the moves. These statistics give us, for each piece (e.g. the c2 pawn) and square, how likely it is to end the game on that square, and how likely it is to be alive or dead there when the game ends [5].

`safe`. This strategy moves pieces towards squares where they are likely to end the game alive. For this strategy and several others, simply moving to maximize this score (e.g. its sum or product over all pieces) is very boring, since the score is almost always maximized in the starting position. So this strategy actually makes each move randomly, weighted by the total score of the resulting positions. The scores are normalized (with the lowest-scoring move receiving weight 0.0 and the highest 1.0) and then sampled according to these weights. Without normalization, the play is almost identical to uniformly random, since the weights of the resulting positions tend to be very similar (dominated by the many pieces that don't move). But it's pretty arbitrary. (state)

`popular`. Like `safe`, but the score for a piece/square pair is the probability that the piece ends on that square, whether it lives or dies. This player likes to follow the crowd! (state)

`dangerous`. The dual of `safe`; the score is the probability that the piece dies on that square. Note that a king is said to die if it is checkmated or his side resigns. This player likes to live life on the edge! (state)

`rare`. The dual of `dangerous`; the score is one minus the probability of ending the game on that square. This player has a thirst for adventure! (state)

`survivalist`. Like the above, but the score is the ratio of the survival and death probabilities on the square. In the data set, every piece ends alive or dead in every square (except for the bishops, which can only legally occupy squares of their color) at least 1000 times, so each ratio is defined. Here, the sums of ratios have plenty of variability, and the highest ratios are not so often on the starting squares. So with this strategy, we simply do a weighted sample from the moves according to their (non-normalized) scores. (state)

fatalist. The dual of `survivalist`; the score is the ratio of the death and survival probability on the square. This player knows that even if you win, you just have to keep playing over and over again, so you might as well get it over with! (state)

2.3 Engine-based players

Of course, people have been making more serious attempts at automating chess since before computers, and there are thousands of chess engines out there. We include a few in here to represent the high end of skill, and to make sure that weaker players are evaluated somewhat in terms of their ability to play chess proper, not just beat other weak players.

stockfish0. Stockfish [6] is probably the strongest open-source chess engine (or even publicly available engine); at full strength its play is estimated to be around 3500 Elo on the FIDE scale. Aside from being quite machine-dependent (it can search more moves in a given amount of time when it has a fast CPU with many cores), there are many options to fiddle with. Stockfish can use both opening books and endgame tables; neither is used here. It also has a "difficulty" setting, which is set here to 0. Stockfish is run in a separate process, and the board is reset before each move, but I am not extremely hygienic about flushing e.g. internal hash tables between moves. One consequence of this attempt at statelessness is that Stockfish sometimes walks into a draw by repetition in positions where it would be able to win, because it doesn't know that it is repeating positions. (trad)

stockfish5. Stockfish as above, but at difficulty 5. (trad)

stockfish10. Same, at difficulty 10. (trad)

stockfish15. Same, at difficulty 15. (trad)

stockfish20. And difficulty 20, the maximum. Even at this difficulty, Stockfish produces moves basically instantaneously. (trad)

stockfish1m. As expected, the engine's performance increases steadily as the difficulty setting increases (without apparently affecting the time to make moves). I don't know what it's doing with these settings. The true way to unleash chess engines is to give them a lot of CPU and memory to search. Since the tournament is run simultaneously across about 60 threads[3] using dozens of gigabytes of memory, and sometimes I would play *Hollow Knight* (*aka* ♘) while it ran, I wanted to avoid having the chess skill be dependent on the scheduling environment. So here, Stockfish is given a "node" budget of 1 million, hence `1m`. It takes about one second per move when running alone, and is easily the strongest player (type) evaluated. (trad)

worstfish. On the other hand, a strong chess engine can also be used to play badly. When playing as white, for each legal move, I ask Stockfish (configured as `stockfish0`) to evaluate the resulting position from black's perspective.[4] I then choose the move that produces the best position for black. This is easily the worst player evaluated, but it is not hard to imagine ways it could be worse. Indeed, a common twist on chess is to play for a loss, called *Antichess* or *Losing Chess* [9]. Recently it was even proved that white can always win (i.e. lose) [8] in this variant! However, the variant requires that you capture a

[3] The computer is the completely egregious AMD 2990WX "Threadripper 2," which has 32 cores (for 64 hardware threads) and 250 Watts of power dissipation at load. The torture of this CPU was part of the impetus for the paper.

[4] By asking it to make a move, which also returns the evaluation of its move. The UCI protocol does not seem to offer a way to evaluate a position directly.

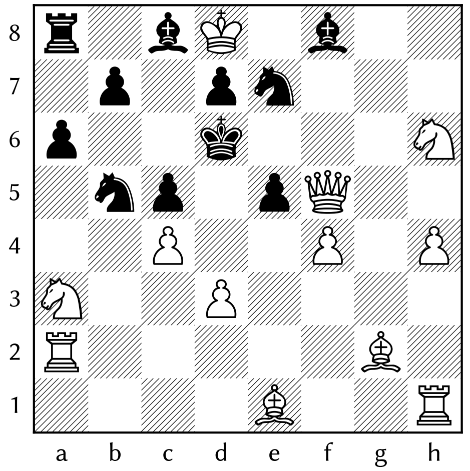


Fig. 8. `suicide_king` (white) to move against `worstfish` after 36 moves. Since the kings are already at their minimum distance, white will make a move at random. 37. Qxe5++ wins for white immediately, but `suicide_king` plays 37. Qxd7??. The only legal move is 37…Bxd7++, so `worstfish` must play it, and thus wins one of its only victories.

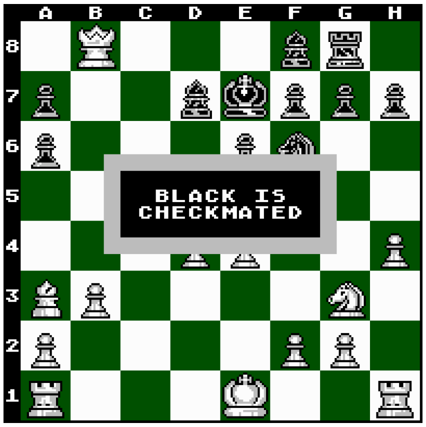
piece if you are able to, so strategies and engines that support this variant cannot be directly applied. We could use stronger settings of Stockfish, but since it already invokes Stockfish for each legal move, it is also one of the slowest players. Like Stockfish, it occasionally blunders an otherwise losing position into a draw by repetition. But most importantly, its search strategy when evaluating positions is not applying the correct logic (α–β pruning); it assumes that its opponent will choose strong moves, and that it will itself play strong moves in future plies. As a result, it sometimes allows the opponent to create a situation where `worstfish` is forced to checkmate its opponent (Figure 8).

`topple10k`. Topple [7] is another strong open source engine, provided to keep Stockfish on its toes. Here, its node budget is 10,000. trad

`topple1m`. Topple with a node budget of 1 million, which like `stockfish1m` takes about one second per move. Stockfish almost always wins, though it is not clear whether the budgets are actually comparable. trad

`chessmaster.nes_lv1`. This is *Chessmaster* for the Nintendo Entertainment System, published in AD 1989. Moves are extracted from the game via emulation. This proved to be somewhat delicate, because the in-memory representation of the board is not that simple (it appears to be represented in two parallel arrays, perhaps using the "0x88 method") and the game understandably goes wild if you mess it up. To play a move, I restore a saved machine state in the "board editor" mode, and then modify memory to contain the desired board. I then emulate button presses to operate the game menu and return to "playing" mode. Chessmaster repairs its internal data structures from this board, and makes a move. During this time I mash controller buttons to skip its dramatic tunes and modal messages like `CHECK`. Once some memory locations reach certain values, the move has been played; I can diff the before and after boards to uniquely determine the move. Since this approach uses emulation, it would normally be completely deterministic, but I deliberately stall for a random number of frames so that it can advance its internal pseudorandom state. A nice thing about this engine is that it is an earnest attempt at writing a good engine, but limited to the techniques of the 1980s, and running on hardware that was underpowered even for its day. It finishes well behind the modern engines, but ahead of all the non-traditional strategies. trad

chessmaster.nes_lv2. As above, but increasing the "Level Of Play" to "Newcomer 2." Chessmaster has stronger levels still, but in these modes it will think for several NES minutes, which takes multiple seconds to emulate—too slow for our purposes. Still much weaker than modern engines (Figure 9). trad



BLACK IS CHECKMATED

Fig. 9. stockfish1m (2019; running on a modern machine) beats chessmaster.nes_lv2 (1989; running on an emulated NES) thousands of times without a loss or draw.

2.4 Blind players

blind_yolo. This player is only allowed to see the 64-bit mask of where pieces are placed on the board, but not their colors or types. It is described in *Color- and Piece-blind Chess* [4].

blind_kings. As blind_yolo, but forcing the prediction to produce exactly one king for each side, which improves its accuracy a little.

blind_spycheck. As blind_kings, but performing a "spy check" before each move. Here, the player tries capturing each piece of a given (predicted) color with other pieces of that same (predicted) color. If such a move is legal, then one of the two pieces was mispredicted, so we prefer the capture over sending an incorrect position to the engine.

Each of these uses a neural network to predict the configuration of the board, and then the equivalent of stockfish1m to make a move for that predicted board. (If that move is illegal because the board was mispredicted, then it plays a random legal move.) These players can therefore be seen as handicaps on stockfish1m.

2.5 Interpolation methods

Even at its weakest setting, stockfish0 crushes all of the nontraditional strategies. This is not surprising, but it creates a problem for quantifying their difference in skill (do they win one in a million games? One in 10^{100}?). Having intermediate players allows some Elo points to flow between the two tiers transitively. One nice way to construct intermediate players is by interpolation. We can interpolate between any two players,[5] but it is most natural to interpolate with the most canonical player, random_move.

stockfish1m_r*nnn*. There are 15 players in this group, each characterized by a number *nnn*. Before each move, we generate a random 16-bit number; if the number is less than *nnn* then we play a random move and otherwise, a move as stockfish1m. The values of *nnn* are chosen so that we mix a power-of-two fraction of noise: stockfish1m_r32768 blends half random with half strong moves, but we also have 1/4 random, 1/8 random, 1/16, 1/32, …, 1/1024. These give us a nice smooth gradation at high levels of play (Figure 11). At the extremes, many games

[5] Even if they are stateful! The interface separates "please give me a move for the current state" and "the following move was played; please update your state," allowing them to disagree. So we can just keep two independent states for the two players.

have no random moves, and the performance is difficult to distinguish from stockfish1m. Even at half random moves, stockfish1m_r32768 consistently beats the non-traditional players. So we also dilute stockfish by mixing with a majority of random moves: stockfish1m_r49152 is 3/4 random, and we also have 7/8, 15/16, 31/32, and 63/64. At this point it becomes hard to distinguish from random_move.

Note that playing 1/64 excellent moves and the rest randomly doesn't do much to help one's performance. On the other hand, playing one random move for 63/64 strong ones does create a significant handicap! Against strong opponents, it's easy to see how a mistake can end the game. Even against weak ones, a blunder can be fatal (Figure 10).

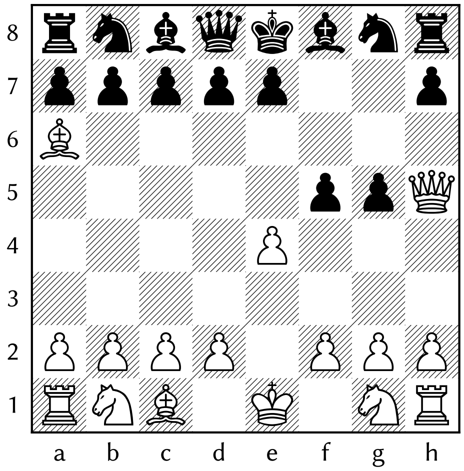


Fig. 10. A demonstration of how quickly random play can ruin a game. stockfish1m_r32768 (black; plays half random moves and half strong moves) loses to reverse_starting in one of the shortest possible sequences: 1. e4 g5 2. Ba6 f5 3. Qh5++. Both of black's moves are bad (so they must be random) but white's 2. Ba6 is a blunder as well. White is just pushing pieces as far as possible; the mate is purely accidental.

3 RUNNING THE TOURNAMENT

Given the players, it's a simple matter to simulate a bunch of games. Since the tournament runs for tens of thousands of CPU hours, there is of course some code to allow it to checkpoint its work, to make efficient use of CPU and RAM, and to provide attractive ANSI graphics showing off its activity—but this is all straightforward.[6] The tournament accumulates win/loss/draw counts for each pair of players (playing as both white and black), as well as example games used to debug or create the figures in this paper.

After the tournament is done, or from any checkpoint, I can run a separate program to produce Elo (and other) rankings. Elo depends on the order in which games are played, because it is designed to take into account changes in player skill over time. However, it is easy to just make up a random order in which the games were played, because these players do not change over time.

Despite its strong reputation, I found Elo to be rather finnicky. It is sensitive to the starting scores for the players, and the factor k which governs how strong of a correction to make after a game. With players of such vastly different skill, it is also very sensitive to the order in which the games are played.[7] For similar reasons, it is also very sensitive to imbalance in the number of games played between a particular pair of players; if some player mostly has games against weak players, this can artificially inflate its score.

[6] The source code can be found at sourceforge.net/p/tom7misc/svn/HEAD/tree/trunk/chess/

[7] For example, suppose the weak player random_move has one win against stockfish1m. If this win happens early, when the two players have their initial Elo scores, then it doesn't affect anything. If it happens last, when stockfish1m has thousands of Elo points over random_move, then it produces a very strong correction.

To control for these effects, I compute the maximum number n for which we have n games played between every pair of players. I then seed each player with an initial Elo score of 1000, and sample exactly n games to play from each cell without replacement. This is so that we play a balanced number of games. I also randomize the global order of all games. Then I do this over again, with a smaller update factor k, for 20 passes. I don't know whether it's the fact that I reduce k over time, or that I end up with effectively 20 times the number of games played,[8] but doing this significantly reduces variance. I run the whole thing 19 times and the result is the median Elo score. I also computed the 25th and 75th percentiles, which were within 1–2% of the median, which is good. The Elo scores appear in Section 4; $n > 400$ for this run.

Ideally, such a system would be robust against adding new players, but this is probably not the case; it is easy to see from the results (Figure 11) that there are certain matchups that favor one of the players despite its average skill. During development, I often noticed significant swings in Elo scores as players were added, especially before there were middle-tier players in the mix. One way to deal with this would be to run the Elo simulations multiple times while randomly ablating players from the tournament; we could then at least estimate the variance in the Elo scores under the removal of players, which is like adding players in reverse. I did not implement such a thing because of oppressive SIGBOVIK deadlines.

4 RESULTS

The main use of the Elo World tournament is to benchmark some new engine or algorithm for playing chess, particularly if it is not that good [4]. There are a few ways that we can interpret the results:

The **Elo score**, as described.

We can find a **comparable interpolated player**. This is like the Scoville scale for measuring how spicy something is: Some golden-tongued blind tasters are given the pure chili oil and asked to distinguish it from a cup of sugar water, which they of course can do. They then dilute the chili oil 1:1 with sugar water, and try again. The number of dilutions before the testers cannot reliably tell the drink from sugar water yields the score on the Scoville scale. Here, for example, we can say that `blind_spycheck` performs between `stockfish1m_r63488` and `_r61440`, so it is approximately a 93.75–96.875% diluted Stockfish.

We can compute a **Markov probability**. The tournament table can be easily thought of as a Markov transition matrix. Imagine that there is a Champion trophy, held by the player corresponding to some row. Each cell in that row contains the probability that in a game betweeen those two players, the trophy would be passed to that player. We treat draws as choosing one of the two sides by a coin flip (or, equivalently, splitting the trophy in half); and for the games that the row player wins, the trophy is retained (probability mass assigned to a self-transition). It is easy to compute this matrix from the win/loss/draw counts in the tournament table, and it is not sensitive to imbalance like the Elo calculation is. Presented this way, we can compute the stationary distribution (if it exists, which it typically will), which basically tells

[8] To be clear, running 20 passes means that games can be reused, which is not ideal.

us that after an extremely large number of games, what is the probability that a given player holds the Champion trophy? This tends to agree with the Elo score, but there are a few places where it does not (e.g. same_color has a much higher p(Champion) score than its Elo seems to warrant).

And so finally, a table with numbers:

Player	Elo	p(Champion)
worstfish	188.54	0.00000071
pacifist	286.90	0.00000509
alphabetical	320.91	0.00000276
generous	342.71	0.00000306
popular	349.48	0.00000374
dangerous	349.60	0.00000291
safe	350.64	0.00000454
first_move	357.71	0.00000434
rare	361.39	0.00000323
no_i_insist	378.23	0.00000244
huddle	399.55	0.00000933
sym_180	429.94	0.00000331
same_color	430.90	0.00002590
opposite_color	432.11	0.00000293
sym_mirror_x	438.87	0.00000305
survivalist	439.03	0.00000681
random_move	439.21	0.00000462
fatalist	439.89	0.00000314
sym_mirror_y	442.50	0.00000716
reverse_starting	445.56	0.00000301
suicide_king	447.02	0.00000493
stockfish1m_r64512	462.82	0.00000297
stockfish1m_r63488	482.44	0.00000356
blind_yolo	488.97	0.00000488
blind_kings	501.98	0.00000633
equalizer	504.21	0.00000653
stockfish1m_r61440	521.45	0.00001173
swarm	534.03	0.00000623
blind_spycheck	546.91	0.00002554
cccp	553.54	0.00000544
min_oppt_moves	597.04	0.00001076
stockfish1m_r57344	600.54	0.00001112
stockfish1m_r49152	752.22	0.00000737
chessmaster.nes_lv1	776.15	0.00002055
stockfish1m_r32768	976.20	0.00003025
chessmaster.nes_lv2	989.21	0.00012094
stockfish1m_r16384	1277.73	0.00012509
stockfish0	1335.69	0.00008494
stockfish5	1644.63	0.00070207
stockfish1m_r8192	1690.15	0.00152052
topple10k	1771.65	0.00179298
stockfish10	1915.23	0.00257648
stockfish15	1952.93	0.00310436
stockfish1m_r4096	2020.25	0.00886928
stockfish20	2139.47	0.00721173
topple1m	2218.71	0.01091286
stockfish1m_r2048	2261.50	0.03132272
stockfish1m_r1024	2425.72	0.06759788
stockfish1m_r512	2521.78	0.11122236
stockfish1m_r256	2581.97	0.15364889
stockfish1m_r128	2609.45	0.17861429
stockfish1m_r64	2637.78	0.20508090
stockfish1m	2644.10	0.21523142

5 CONCLUSION

Shout out to the Thursd'z Institute and anonymous commenters on my blog for discussion of players and suggestions. Several ideas were suggested by multiple people, increasing my confidence that they are somehow canonical.

The author would also like to thank the anonymous referees of the Special Interest Group on Bafflingly Overdone Ventures In Chess journal for their careful reviews.

REFERENCES

[1] 2017. FIDE Handbook – B.. Permanent conditions. www.fide.com/component/handbook?id=2.

[2] Arpad E Elo. 1978. *The rating of chessplayers, past and present.* Arco Pub.

[3] Amanda M. Miller and David L. Farnsworth. 2013. Counting the Number of Squares Reachable in k Knight's Moves. *Open Journal of Discrete Mathematics* 03, 03 (2013), 151–154. https://doi.org/10.4236/ojdm.2013.33027

[4] Tom Murphy, VII. 2019. Color- and piece-blind chess. In *A Record of the Proceedings of SIGBOVIK 2019.* ACH.

[5] Tom Murphy, VII. 2019. Survival in chessland. In *A Record of the Proceedings of SIGBOVIK 2019.* ACH.

[6] Tord Romstad, Marco Costalba, and Joona Kiiski. 2019. Stockfish Chess. https://stockfishchess.org/.

[7] Vincent Tang. 2019. Topple. https://github.com/konsolas/ToppleChess/releases/.

[8] Mark Watkins. 2017. Losing Chess: 1. e3 Wins for White. *ICGA Journal* 39, 2 (2017), 123–125.

[9] Wikipedia. [n. d.]. Losing Chess. http://en.wikipedia.org/wiki/Losing_Chess.

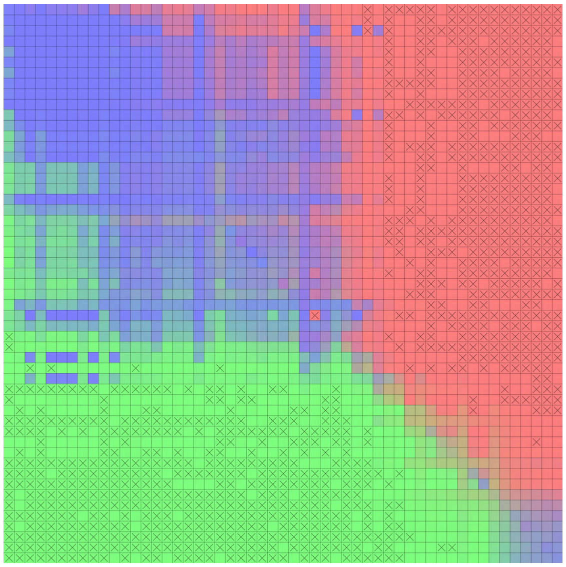


Fig. 11. The matrix of outcomes for all players. There is too much data to print, so we just have a bunch of colors, mostly because it looks rad. If you don't have a color printer or TV it will probably not look rad. Rows indicate the player playing as white, from worst (`worstfish`, top) to best (`stockfish1m`, bottom). Columns for the player with the black pieces, in the same order from left to right. Green indicates a cell that is predominately wins (for white), red is losses, and blue is draws. The right and bottom third of the graphic are all different levels/dilutions of strong engines (Stockfish and Topple). This creates a nice smooth gradient where better engines regularly beat even slightly weaker ones, but start to produce draws at the highest levels. They consistently destroy weak engines; a cell with an × indicates that it only contained wins or only contained losses (not even draws).

At the low end, there is a large splat of matchups that mostly produce draws against one another, but can consistently beat the weakest tier. In the top-left corner, a square of blue draws from low-ambition play; these aggressively bad players almost never win games, even when playing each other.

Microtexture outside of these broad trends comes from matchups where the player is unusually suited or weak against that particular opponent. For example, the bright red cell on the diagonal near the center is `cccp` vs. `cccp`; this determinisic strategy alwasy wins in self-play as black.

A Formal Treatment of $^{k}/_{n}$ Power-Hours

Christian Clausen

1 Related Work

Power Hour is the name of a drinking game in which you take one shot of beer every minute, and while this is fun, sometimes you may prefer to drink slower. This requires a generalization of the Power Hour, however sophisticated algorithms for this are prone to memory leaks. This problem is addressed in [1] with a category of algorithms called $^{k}/_{n}$ Power-Hours.

Further studies of the Power Hour were made in [2] where the author correctly points out that more games were possible then reported in [1]. He further studies the effects of adding multiple players.

We present here a pictorial representations to address further memory leaks, a generalization over states, along with a series of lemmas that suggest an extra possiblity not considered earlier in the $^{k}/_{n}$ Power-Hours. We also include an appendix containing graphs over all the possible fair 4 state, one person games.

2 Introduction

Let's start with some notation. We have adapted the notation of [1] where $\uplus$ means full, $\cup$ means empty, and $\cap$ means flipped. Further we have extended it with the symbol $+$ to shorten: fill and drink.

The rules of a $^{k}/_{n}$ Power-Hour are as follows: every minute you have to perform one or more actions ending in a (possibly new) state. Due to memory concerns the actions should be uniquely determined based on the current state of your glass. If you have a full glass you have to drink it, before executing the actions. You will end up drinking k shots in n minutes.

Now we are equipped to take a first look at our pictorial representation of a configuration, specifically the $^{2}/_{60}$ as described by [1]. [1]

$$\begin{tikzcd}
\uplus \arrow[rr] & & \cup \arrow[dl, "+"] \\
& \cap &
\end{tikzcd}$$

Figure 1: $^{2}/_{n}$

It reads: on $\uplus$ empty it and leave $\cup$. On $\cup$ fill and drink, then leave $\cap$.

3 A Multitude of States

In this section we present our generalized lemmas, along with examples.

The first generalization we need to discuss is the number of states, while the conventional 3 are neat we suggest a fourth $\subset$ (lying), that can be used without conflict with the others. How-

[1] We usually omit identity arrows, as here from $\cap$ to $\cap$, unless we only use one state.

ever we advice against using it together with $\supset$, as this will cause confusion.

Having offered an extra state we can only assume other people will do so as well. Therefore we will write ${}^{k}\!/^{s}_{n}$ for a ${}^{k}\!/_{n}$ game with s states. If nothing is indicated we assume a 3 state game.

Definition 1. a fractional game is a ${}^{k}\!/^{s}_{n}$ where k is some fraction $\frac{f(n)}{d}$. A *pure* fractional game has $s = d$.

Specifically this means a game with a graph where there is a cycle, the longest circle is d steps, and $\uplus$ is in it.

Lemma 2. *for a* ${}^{k}\!/^{s}_{n}$ *power hour game there are* s *trivial fractional configurations;* ${}^{\frac{n}{1}}\!/_{n}, {}^{\frac{n}{2}}\!/_{n}, \dots, {}^{\frac{n}{s}}\!/_{n}$.

Proof. by induction on the number of states. □

With the conventional 3 states they look like this:

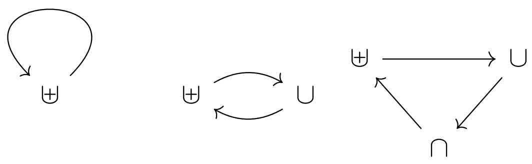


Figure 2: ${}^{\frac{n}{1}}\!/_{n}$, ${}^{\frac{n}{2}}\!/_{n}$, ${}^{\frac{n}{3}}\!/_{n}$

An interesting aspect is that we can add + to any of these arrows, the effect of which is are presented in the following lemma:

Lemma 3. *for a fractional* ${}^{\frac{kn}{d}}\!/^{s}_{n}$ *game with* $k \leq d$, *can make it into a* ${}^{\frac{(k+1)n}{d}}\!/^{s}_{n}$ *game.*

Proof. we add 1 drink pr. d steps, in a k step game, thus we have $\frac{1}{d}n + \frac{kn}{d} = \frac{n}{d} + \frac{kn}{d} = \frac{n+kn}{d} = \frac{(k+1)n}{d}$. □

We can now make the more interesting ${}^{2n}\!/_{n}$ and ${}^{\frac{2n}{3}}\!/_{n}$ games;

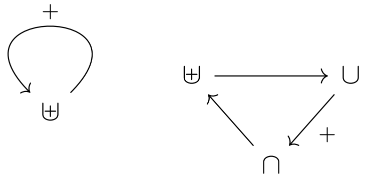


Figure 3: ${}^{2n}\!/_{n}$, ${}^{\frac{2n}{3}}\!/_{n}$

In fact;

Theorem 4. *all possible pure fractional games* ${}^{\frac{kn}{d}}\!/^{s}_{n}$ *can be constructed from the previous two lemmas.*

Proof. left as an exercise to the reader. □

Corollary 5. all possible pure fractional games can be written on the form ${}^{\frac{kn}{s}}\!/^{s}_{n}$, with $k \leq s + 1$.

4 Constant Amount Games

In this section we look at ${}^{c}\!/^{s}_{n}$ games where c is not affected by n. Intuitively this means that we cannot have (non-identity) cycles in the graphs. We already saw the ${}^{2}\!/_{n}$ game. However which games are possible? In [1] the authors claim that a ${}^{3}\!/_{60}$ game is impossible[2], however we claim that it is possible, and even stronger:

Lemma 6. *with* s *states, and* $n \geq s$ *there are* $s + 1$ *lower constant games:* ${}^{0}\!/^{s}_{n}, {}^{1}\!/^{s}_{n}, \dots, {}^{s}\!/^{s}_{n}$.

Proof. by induction on the number of states. □

[2] they were conducting relevant research while writing it, so it is understandable.

Continuing with the classical examples they are[3]:

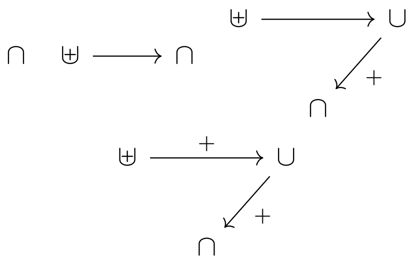


Figure 4: $^{0}/_{n}, ^{1}/_{n}, ^{2}/_{n}, ^{3}/_{n}$

At this point you may think that we are done with constant games, however as noted by ... there is some symmetry in this game, and we can take advantage of this here;

Lemma 7. *with s states and $n \geq s$ there are s upper constant games: $^{n-s+1}/^{s}_{n}, ^{n-s+2}/^{s}_{n}, \ldots, ^{n}/^{s}_{n}$.*

Proof. by induction on the number of states. □

You may have already figured out how they look, but here they are;

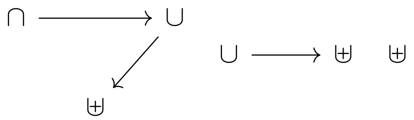


Figure 5: $^{n-2}/_{n}, ^{n-1}/_{n}, ^{n}/_{n}$

5 Having a Party

Definition 8. a game $^{k}/^{s}_{n}$ with $k \in \mathbb{N}$ is called fair.

Now it is time to mix it all together:

Theorem 9 (Soundness)**.** *there is a fractional $^{\frac{kn-c}{d}}/^{s}_{n}$ game if $k \leq d+1, d+|c| \leq s$, and $kn-c \geq 0$.*

The equation may look complicated, but it is quite logical when you look at a picture:

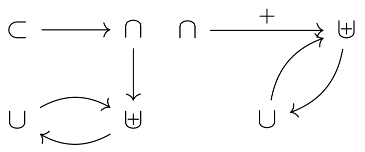


Figure 6: $^{29}/^{4}_{60}$, $^{\frac{59}{2}}/_{60} \cong {}^{31}/_{60}[2]$

And finally we are ready to express the master lemma:

Theorem 10 (Completeness)**.** *all possible fractional games can be written as $^{\frac{kn-c}{d}}/^{s}_{n}$ with $k \leq d+1, d+c \leq s$ and $kn-c \geq 0$.*

Which captures all the intuitions we have built.

Now that we have that in order, should invite some friends:

Lemma 11. *if $d-1$ players take turns playing with the same cup, and there is a fair game $^{\frac{kn-c}{d}}/^{s}_{n}$ with $p \mid c$ then there is a fair $^{\frac{kn-c}{pd}}/^{s}_{n}$ game.*

This means that with 2 girls, 1 cup, and an hour the possible games are:

$$^{\frac{60k}{2\cdot 3}}/^{3}_{60} = {}^{10k}/^{3}_{60}$$

for $k \leq 4$ (due to proposition 5), giving the new configuration $^{10}/_{60}$. If we also have a die we could even get: $^{10k-1}/^{9}_{60}$ for $k \leq 4$. And with 3 people we could play $^{5k}/^{4}_{60}$ for $k \leq 5$, giving the new (fair) games: $^{5}/^{4}_{60}, ^{25}/^{4}_{60}$.

[3] Notice that we end on a ∩ to indicate that it does not need refilling.

6 Bibliography

[1] Ben Blum, Dr. William Locas, Chris Martens, Dr. Tom Murphy VII, *Algorithms for k/n Power-Hours*, SIGBOVIK 2012

[2] Dr. Tom Murphy VII, *New results in ${}^{k}\!/_{n}$ Power-Hours*, SIGBOVIK 2014

A 4 State Games

A.1 Trivial Fractional Games

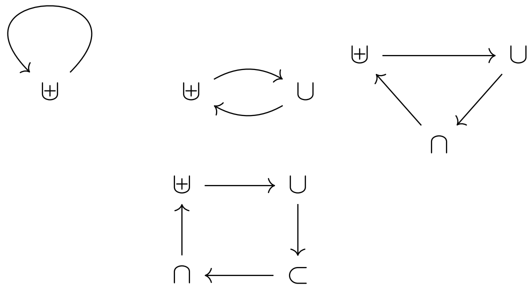


Figure 7: ${}^{\frac{n}{1}}\!/^{4}_{n}$, ${}^{\frac{n}{2}}\!/^{4}_{n}$, ${}^{\frac{n}{3}}\!/^{4}_{n}$, ${}^{\frac{n}{4}}\!/^{4}_{n}$

A.2 Other Fractional Games

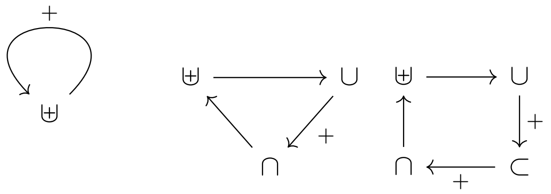


Figure 8: ${}^{2n}\!/^{4}_{n}$, ${}^{\frac{2n}{3}}\!/^{4}_{n}$, ${}^{\frac{3n}{4}}\!/^{4}_{n}$

A.3 Lower Constant Games

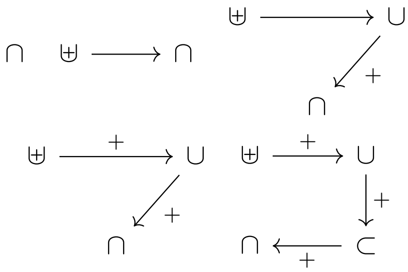


Figure 9: ${}^{0}\!/^{4}_{n}$, ${}^{1}\!/^{4}_{n}$, ${}^{2}\!/^{4}_{n}$, ${}^{3}\!/^{4}_{n}$, ${}^{4}\!/^{4}_{n}$

A.4 Upper Constant Games

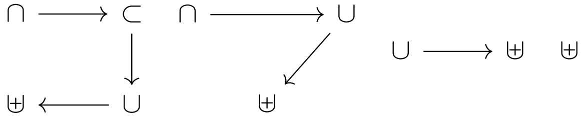


Figure 10: ${}^{n-3}\!/_{n}$, ${}^{n-2}\!/_{n}$, ${}^{n-1}\!/_{n}$, ${}^{n}\!/_{n}$

A.5 Mixed Games

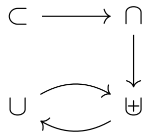


Figure 11: ${}^{\frac{n-2}{2}}\!/^{4}_{n}$

Eventually Consistent Partying

Veit Heller

March 9, 2019

Abstract

In distributed systems and life in general, eventual consistency is the desirable state of agreement. In this paper, we show how to reach this state in the context of parties with regards to the buzz factor (also known as inebriation quotient).

Keywords Party System Design, Buzz Factor, SIGBOVIK

1 Introduction

In classical distributed systems, eventual consistency is a state of agreement between nodes in a system that is reached at a certain point in time. This property is usually desirable because it provides clear grounds on which to base assumptions about the state of the system.

At parties, too, there is a certain state of agreement—or agreeability, if you will—that is usually beneficial to the mood of the actors in the system, a property worth optimizing for. This property is in strong correlation with the buzz factor—also known as inebriation quotient—, which describes the state of alcohol saturation of a given actor.

In this paper, we describe a novel approach to reason about eventual consistency as it relates to partying.

2 Preliminaries

Our foremost goal for eventual consistency in a party setting is reaching a quorum of approximate buzz level. It is said to be almost impossible to set up a perfectly consistent system unless you set up a group of exclusively sober actors. There is no consensus on whether a group of sober actors can be classified as a party in the literature, as some authors prefer to call those systems "tea parties".

We characterize an actor by their buzz intake behaviour and their buzz threshold. If the buzz threshold is reached, we characterize an actor as having dropped out and disregard them when checking for a quorum. The buzz threshold seems to be correlated to the actor's gender, age, and bodyweight.

At this stage, it is important to note that there is no standard unit that allows us to talk about buzz-related activities. For the purposes of this paper, we will assume the standard unit to be "one can of beer (Fat Tire Amber Ale)". Interested parties suggested the authors use the more well-known unit of "one can of beer (Bud Light)", but that idea was quickly discarded because the authors had doubt that a credible party could be based on said unit.

The sole medium of buzz accounted for in this paper is ethanol. While other substances seem to be in use for party-related buzzing, describing their respective interrelations would complect the calculations tremendously while accounting only for a reasonably small subset of actors.

2.1 Archetypes

In our research we discovered four primary archetypes of actors at parties in relation to attaining the buzz factor. We will describe them in the following.

Serious Business. Actors in the "Serious Business" class—also known as "winos" or "boozehounds"—are characterized by a quick intake of buzz. Depending on their buzz threshold, they might drop out early.

Dead Sober. Actors in the "Dead Sober" class—also known as "buzzkills" or "designated drivers"—do not intake buzz. Accordingly, they either have to be in the majority to achieve what is known as an *a priori* quorum, or be in the minority, in which case they will work against the quorum of the other actors.

Low and Slow. The "Low and Slow" class of actors—also known as "normies"—shows a slow, steady intake of buzz. Any spikes in buzz may be counteracted by the intake of water or other non-alcoholic substances, though the increase of buzz over time is almost inevitable.

Sugar Rush. Favoring Cocktails and longdrinks, this class of actors—also known as "fancypants" or "amateur baristas"—show almost no signs of buzz for a period of time—this period is unpredictable and largely based on the compounds used when intaking buzz—before a sudden spike of buzz that can easily overshoot the buzz threshold. That point in time is known as "the tipping point" in the literature, and will be referred to as such in the following.

While this collection of archetypes is helpful in building a vocabulary for expressing actor behavior, it should not be taken as canonical or exhaustive. Party architecture is an understudied area of systems design, and we expect many novel, more accurate categorizations to emerge in the coming years.

3 Practical Discussion

Now that we have laid the groundwork to conceptualize the system, we will develop a vocabulary of basic buzz-related calculations.

3.1 Calculating the Buzz Factor

We have found that a reasonable way to calculate the buzz factor is to presume that there exists a function `f` that, given a time `t`, returns the buzz intake at that time. Calculating the total buzz factor at any given point in time is then as simple as taking the sum of buzz intakes from `t0`—being the time at which the party started—to `t1`—being the time for which the buzz factor should be calculated.

When accounting for each actor independently, it is important to adjust `f` based on their archetype. The model of the archetype might need to be adjusted slightly to account for the individual differences between actors within the same archetype category. Machine Learning algorithms might be able to predict better versions of `f` for an actor given behavior samples during previous parties, though that area has heretofore been vastly understudied.

In the following we shall give an overview of functions that seem to best predict actor behavior per archetype. Note that the buzz threshold cannot be predicted and needs to be measured or assumed by the architect.

Serious Business. Actors in this class should best be modelled by a function that returns a static value, since their buzz factor is usually monotonically increasing. An increase by one unit of measurement per hour can usually be assumed. When accounting for individual actors, that number might change, also in accordance with the buzz factor, since some actors in this class seem to drink more or less depending on their current inebriation quotient.

$$f(t) = 1$$

Figure 1: `f(t)` for "Serious Business".

Dead Sober. The function that describes this class of actors will always return `0`.

$$f(t) = 0$$

Figure 2: `f(t)` for "Dead Sober".

Low and Slow. This class of actors are modelled similarly to actors of the "Serious Business" archetype, with the caveat that, as the party goes on, they might occasionally lower their buzz factor. A base likelihood of `1/3` can be chosen, meaning that once every three units the buzz factor will be reduced rather than increased. This process can be made to seem more "lifelike" by introducing random or even fandom numbers [1]. It is also reasonable to assume a slower buzz intake of about half a unit per hour.

$$f(t) = \begin{matrix} -0.2 & \text{if } t \text{ is evenly divisible by 3} \\ 0.5 & \text{otherwise} \end{matrix}$$

Figure 3: `f(t)` for "Low and Slow".

Sugar Rush. A lot of work on party systems design has been devoted to modelling the actors in this class, and for good reason. While the actor count of this archetype across the entire population seems to be smaller than that of the "Low and Slow" archetype, for instance, all functions modelling their buzz intake have been the subject of a high level of scrutiny in the party architecture community, since none of them have yet seemed to fit a majority of actors. A combination of two positive values, one for the time values before the tipping point, and one for the time values after the tipping point, has led to promising results. The tipping point varies, but an average of four hours into the party seems to be a reasonable assumption.

$$f(t) = \begin{matrix} 0.2 & \text{if} t < 4 \\ 1.5 & \text{if} t \geq 4 \end{matrix}$$

Figure 4: `f(t)` for "Sugar Rush".

3.2 Modelling a Party

To model a party, the current buzz factor of each actor that has not dropped out—i.e. each actor whose buzz factor is below their buzz threshold—must be calculated. A quorum of buzz factors is reached when a majority of actors have approximately the same buzz factor, where approximate means a distance of not more than 1.5 units of measurement between each actor.

3.2.1 Example

Let us model an eight hour long party of four people as an example. The attendees consist of one designated driver of the "Dead Sober" archetype, two normies in the "Low and Slow" category, and one wino who means "Serious Business". There is no actor of the "Sugar Rush" archetype present because the host forgot to buy Ginger Ale and limes.

Choosing a reasonable buzz threshold for each actor, using the standard functions from above, and graphing the results, we end up with a figure that clearly shows that we have a quorum at the hours 1-4, after which point the buzz factors diverge by more than 1.5 units between the designated driver and the normies.

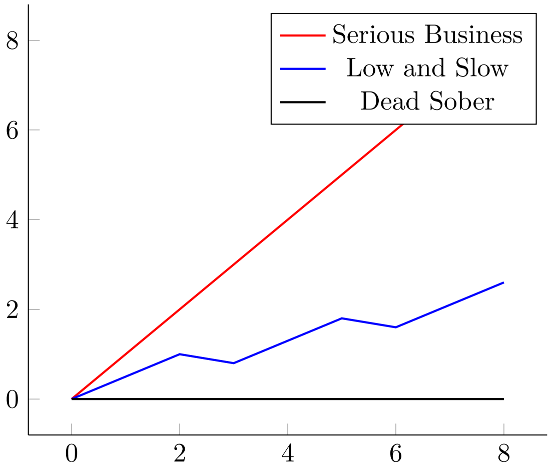


Figure 5: A party, graphed.

4 Conclusion

Party systems design is a nascent, but promising field. We believe that by introducing ideas from the wider world of distributed systems the field can develop a better vocabulary and more rigorous design methods which will eventually lead to a higher quality of parties.

References

[1] Lindsey Kuper and Alex Rudnick. Fandomized algorithms and fandom number generation. In *A Record of the Proceedings of SIGBOVIK 2013*, April 2013.

Survival advice from a computer scientist

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess

Survival in chessland

Dr. Tom Murphy VII Ph.D.[*]

1 April 2019

Abstract

CHESSMATE.

Introduction

If you are forced to play chess to the death, you are in trouble, because most people are not good at chess (for example, the author) and yet want to live.[1]

But what if you are forced to *be one of the chess pieces* to the death? That is, your little soul inhabits one of the 32 pieces or pawns and your soul is vanquished if that piece is eliminated. Now it doesn't matter whether you're good or bad at chess, because you don't get to pick what happens in the game. What matters is that your piece lives to the end of the game, when all surviving pieces are set free. Which piece should you want to be?

In formal chess, the king can never be captured: The game ends when the king is attacked but cannot move, and it is illegal to make a move that leaves the king attacked. The king's death is implied, of course, but it is seen as more poetic to end the game prior to this point.

For the sake of this question, we'll consider the the white king to "die" if white loses (i.e., is checkmated), and likewise for black. Otherwise, of course, the best chances of survival would trivially be with the two kings, since they are never formally captured. Loss includes resignation, since most high-level games actually end once the defeated player agrees that loss is inevitable. We can think of this common case like king seppuku. Many games also end in time forfeit, which is like the king's poor diet and lifestyle choices leading to a death by natural causes.

Neither side is believed to have a decisive advantage, and many games end in a draw, with both kings surviving. So the survival chances of a king are likely greater than 50%; pretty decent odds. Is it possible that any other piece has even better chances? Let's find out—our lives may depend on it!

[*] Copyright © 2019 the Regents of the Wikiplia Foundation. Appears in SIGBOVIK 2019 with the threefold repetition of the Association for Computational Heresy; *IEEEEEE!* press, Verlag-Verlag volume no. 0x40-2A. 53 Centipawns

[1] It is easy for two players to collaborate to produce a draw, especially by simply agreeing to a draw at the outset of the game (if allowed). Some tournament formats forbid the players from agreeing to a draw verbally before a certain point (e.g. 30 moves), or without the arbiter's consent, and FIDE rules technically do not allow a draw until both players have made a move (5.2.3). There are always other routes to a draw, for example by stalemate or repeating the same position three times. Collaboratively producing such situations is easy, but this strategy is not likely a stable equilibrium: Players can often gain a substantial advantage by going "off script" and instead trying to win the game. Additionally, sometimes the terms of chess-to-the-death do not allow the players to communicate at all beforehand, nor during the game. If this is the case, then it may be difficult to agree on the approach to drawing, let alone establish that this is both players' desire. Since the rules of chess-to-the-death can't forbid us from colluding right now as you read this paper, I hereby declare that the following is the correct approach:

1. Nf3. This is a reasonable opening move for white (begins the Réti) which can transpose into several common systems (e.g. King's Indian). Since the knight can move back to g1 on the next move, knight moves are the fastest route to a draw by repetition. This has a good chance of signaling to a wise player that a draw is desired. The player should make this move after pondering carefully for some time, and then looking meaningfully into the other player's eyes.

1. . . . Nf6. This is both a strong response for black in a real game, and simultaneously a signal that a draw is desired. The other advantage is that very weak players[3] may simply copy what white does. In doing so, they will also play this move.

2. Ng1?!. White moves the knight back to its starting square. This is a terrible move for white, but clearly signals the intention to draw.

2. . . . Ng8!. "Fool's Draw Accepted." The starting position is reached for a second time.

3. Nf3 Nf6. At this point the game should clearly continue repeating the sequence, although since we are in the start position, white has any number of strong opening moves available. Signaling the draw line and then 3. d4!? may be pyschologically devastating.

4. Ng1 Ng8 1/2-1/2. The starting position is reached for the third time, which by rule[2] is a draw.

Since this is one of the shortest possible routes to a draw, I hereby dub this line the "Fool's Draw," by analogy with the Fool's Mate. In this line all pieces survive, which is anyway humane and also advantageous in the case that you or someone is simultaneously being one of the chesspieces to the death!

If 1. . . . Nc6 or another Knight's move, white can also consider continuing in the obvious way. However after 1. . . . d5, black has refused or not noticed the draw. Fortunately, white is still in a good position to play the game normally (this is the main line of the Réti opening, followed by 2. c4). White can try to be more obvious with 2. Ng1, but if black is choosing to just play normally, white takes a distinct handicap by doing so.

The biggest risk for white is that black does not play 2. . . . Ng8 but rather a normal move like 2. . . . g6 ("Fool's Draw Betrayed"). This can happen if black is not metagaming at all (for Nf6 is a normal response to the normal Nf3), or if black is an exceptionally shrewd metagamer (tricking white into wasting two tempos with Ng1 by pretending to be cooperating).

Of course, this all relies on the assumption that if chess-to-the-death ends in a draw, the players are spared or allowed to repeat indefinitely. If both players are actually executed, then this line is truly a Fool's Draw!

[2] But is it?

First of all, although either player is allowed to *claim* a draw after three repetitions of the same position, it is not automatic. However, FIDE rules do declare that the game simply ends in a draw upon *five* repetitions. Of course it is easy to extend the Fool's Draw to accommodate this.

Second: The lichess implementation (although known to be buggy[2])

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess

1 Hypotheses

Like all good scientific research, I clearly laid out my hypothesis and wrote down the motivation before performing the study. This helps prevent presentation bias where the results appear more satisfactory because they are framed as a natural conclusion from the idea that motivated the research in the first place (when in fact, of course, if you write the motivation after witnessing the results, backflow is inevitable). It is also much more exciting. I literally don't know the answer as I'm writing this, nor whether it is interesting in any way!

Here are my guesses.

- Black and white are probably not substantially different. That is, the a1 and a8 rooks probably have about the same survival chances (it's known that white has a slight statistical advantage [4] but it is probably only around 1%). So these guesses will be written about white's pieces.

- The d2 and e2 pawns are very active in common openings, and are frequently captured as part of those openings. I think they are the least likely to survive overall.

- Bishops and knights are often involved in the opening and midgame, and often exchanged nonchalantly. I think they all have relatively low survival chances.

- Although the queen is very valuable, a queen exchange is often forced for games that enter the endgame.

- Rooks tend to be late-game pieces, because they are difficult to get out of their corners (and at most one can be activated by the fastest method, castling) and are relatively valuable.

- This leaves the non-central pawns. These are the hardest to predict, and they are hard to think about (at least for me) because when e.g. the a2 pawn recaptures the b3 pawn that it supported, I just think of this as the b3 pawn. Of these pawns, b2 and g2 are somewhat weak because they are undefended once the bishop is developed (cf. the famous "poison pawn" at b2). On the other hand, in the fianchetto configuration, this pawn is very strong and often survives the entire game without leaving the third rank. Since pawn chains usually progress towards the middle of the board, the a2 pawn is more likely to be supporting than supported. This both leaves it weak to capture, but prone to recapturing. Outside pawns block one's own rook, although for this same reason they often clear the file by capturing (and so survive). They are also commonly used to push into a well-defended king's territory (e.g. in the fianchetto); kingside castling is more common, so this means that the h pawns are often lost to this fate.

The final ranking that I predict, from most surviving to most dead: ♙f, ♙c, ♙g, ♙a, ♙h, ♙b, ♖h, ♖a, ♔, ♕, ♗f, ♘g, ♙e, ♙d.

As already copped to, while the author is an aficionado and also knows how to spell the difficult word aficionado without spell-check, he is not good at chess. A few drinking buddies with varying chessperience were also consulted for their wagers; these are compared to each other and to the actual results in Section 4.

2 Methodology

To compute the chances for survival, I legally acquired 506,000,416 chess games from lichess.org. This is all of the standard variant, rated games from Jan 2013 to November 2018, in any time format. Only games that are completed and valid are included (about 200,000 games did not meet this criteria). The total data size is 875 gigabytes, so processing these took some care for efficiency and parallelism. Fortunately, I have a computer with just an obscene number of cores and truly excessive RAM, so you gotta use that for something.[3]

Other than that, I simply implemented the rules of chess, wrote a PGN parser, then parsed and simulated each game. For each of the 32 pieces in the starting position, I tracked its current location, and whether it is alive; multiple dead pieces could occupy the same square. At the end of the game, one of the kings is killed if his side has lost.

For a piece, there is a single factual survival rate in these games, given just by $\frac{\text{num survived}}{\text{num games}}$. What we're really interested in, however, is estimating the underlying true survival probability for each piece. In order to do this with reasonable efficency, we divide the games into 32 separate buckets, and

does not permit a threefold repetition claim in this situation, which got me thinking that maybe there is some subtlety here. Is the starting position special somehow, not counting as having occurred? The relevant statute, from the FIDE Laws of Chess[1]:

> **9.2**. The game is drawn, upon a correct claim by a player having the move, when the same position for at least the third time (not necessarily by a repetition of moves):
>
> a. is about to appear, if he first writes his move, which cannot be changed, on his scoresheet and declares to the arbiter his intention to make this move, or
>
> b. has just appeared, and the player claiming the draw has the move.
>
> Positions are considered the same if and only if the same player has the move, pieces of the same kind and colour occupy the same squares and the possible moves of all the pieces of both players are the same. Thus positions are not the same if:
>
> 1. at the start of the sequence a pawn could have been captured en passant.
> 2. a king or rook had castling rights, but forfeited these after moving. The castling rights are lost only after the king or rook is moved.

So the question is, has the starting position "appeared" before white's first move? The rules are not totally clear on this point. Note that "positions are considered the same" only when the same player "has the move." FIDE defines "have the move" as

> 1.3. A player is said to 'have the move' when his opponent's move has been 'made'.

A strong case can therefore be made that white does not 'have the move' in the formal sense at the beginning of the game, since black has not made a move!

Nonetheless, it does seem clear that white can claim a draw by 9.2.a, by committing the move 5. Nf3 and declaring to the arbiter that the position is now *about to appear* for the third time. This seems unambiguously legal.

[3] To torture your own desktop computer, source code is available at sourceforge.net/p/tom7misc/svn/HEAD/tree/trunk/chess/.

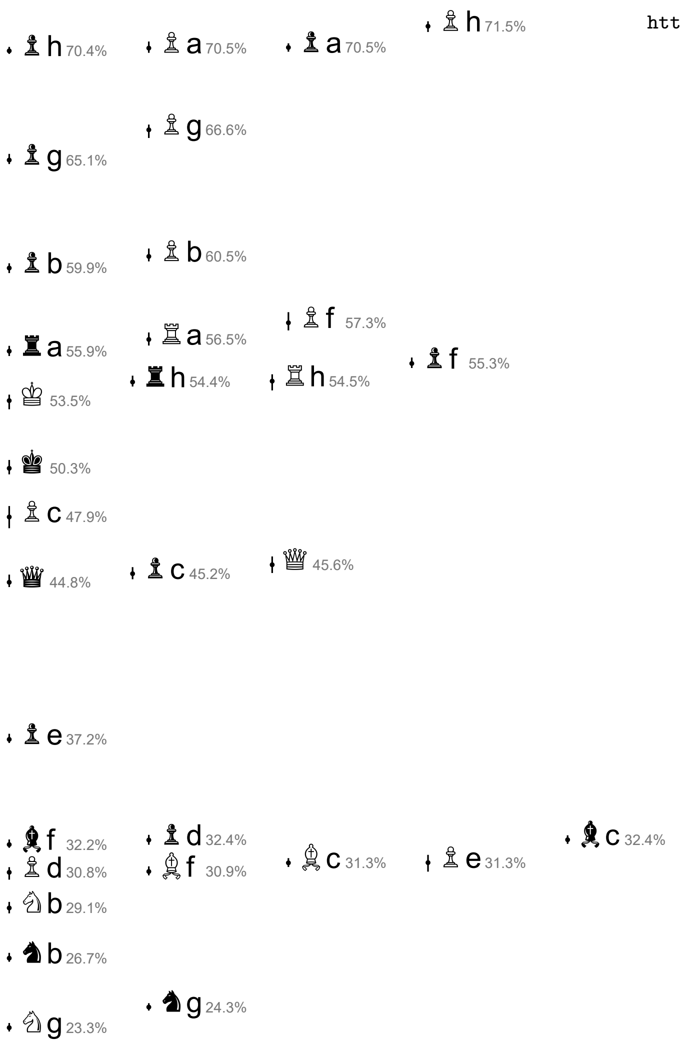


Figure 1: Survival probabilities for each of the 32 pieces in standard chess, in 500 million games. The number is the mean survival rate across all samples. The vertical position of the dot is this mean rate, with a line drawing the span between the smallest and largest sample bucket (this is usually a very tiny range). Horizontal position is purely presentational, to avoid overlap.

count statistics separately for each. From these samples we can then estimate variance, for example. The games are bucketized by a deterministic hash of the White player's username. This way, if there exist some players who are highly unusual (perhaps automated accounts), their games are grouped together and pessimistically represented in the variance estimate. This also helps account for different opening preferences; the chosen opening certainly affects the survival chances.

The basic survival chances appear in Figure 1. Indeed, many pieces are more likely to survive than the kings. Even as black, the extremal pawns (♙a and ♙h) have over a 70% survival rate. Across the board, the survival chances for a white piece and its black twin are similar, usually with a small edge to white. Notable exceptions are the ♘g (the overall most doomed piece), and both white bishops, which die more than their Schwarzdoppelgangers. The ♙e is vastly more dead than ♟e. Note somewhat satisfyingly that the ♙c has the highest variance; this was the most controversial among the g buddies (Section 4). Note that ♙c is the sacrificed the popular Queen's Gambit (1. d4 d5 2. c4), wh oting and declining the pawn are both common and sound responses. This may be a good example of a piece that has substantially different survival rates in different opening preferences. Since the variance is otherwise extremely low, I only report means for the remainder of the paper.

Despite my impression that many games end in a draw, ties are actually rare in the lichess database. In January 2018, only 3.8% of games were drawn; as a result, the survival chances for the kings are both close to 50%. Although the database contains games in many time formats and with all varieties of human skill (including over a thousand games by Magnus Carlsen, the world champion and highest rated player of all time[4]), blitz (∼ 5 minutes per side) and bullet (∼ 1 minute) games are predominant. Nonetheless, the results are fairly robust across different time formats and skill levels. In Section 4.1 I show some slices of the data for comparison.

3 Safest spaces

The fate of each piece is to either survive or die, and it does so on one of the 64 squares. With the same replay of the 500 million games I also kept statistics on the fates of each piece. If being a chess piece to the death, and possessing some influence over where your piece moves, it may be helpful to know where to go. Even without influence, such knowledge could help calibrate your anxiety.

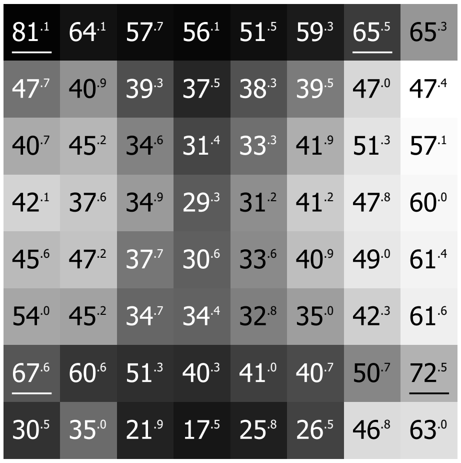
Other than the bishops—which have no legal way to reach half of the squares—every piece ends on every square in at least a thousand games. So we have enough samples to have reasonable confidence in our statistics, even for the most unlikely odysseys. The least mobile pieces are the pawns, who can technically reach any square by promoting, but are usually confined to cones emanating from their start squares. The overall rarest fate is for ♙f2 to die on the a7 square, which only happened 1,244 times (however, it survived on this square 31,438 times). This square is actually reachable without promoting, but it would need to capture 5 times in order to get there, which seems quite unlikely! There may even be a hidden achievement for reaching this square this way! Aside from pawns, the weirdest fate is for ♘g to die on h1, which happened 47,307 times. Corners are of course garbage for the knight, although it is twice as likely to survive on this square.

There are characteristic patterns for each of the pieces, which make sense given their starting positions movement rules. You could probably guess the piece just by looking at one of the heat maps below, although—spoiler alert—the piece is just listed right there and they are in order. Two independent things are communicated in these graphics: The chance that the piece ends the game on some square (alive or dead), and its survival chances there. In each map, a darker background color indicates that the piece ends on that square more often. The shade is based on the rank (64/64 black is most common, 63/64 black is next most common, etc.) rather than

[4] Although to be fair, his username "DrDrunkenstein" suggests he may not play at full strength.

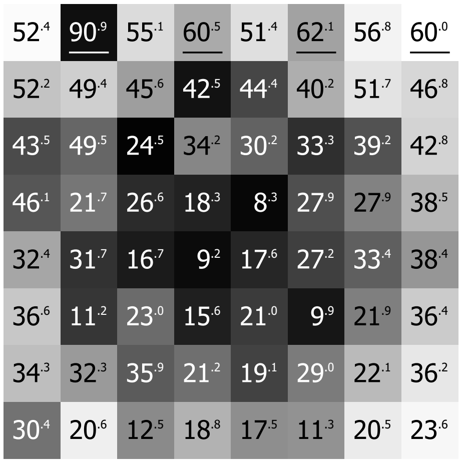
https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess

the absolute probability, since otherwise the graphic looks boring. The number on the square is the percentage survival of the piece when it is last seen (either being captured or surviving to the end of the game) on that square. The four squares with the highest survival rates are underlined for your convenience.



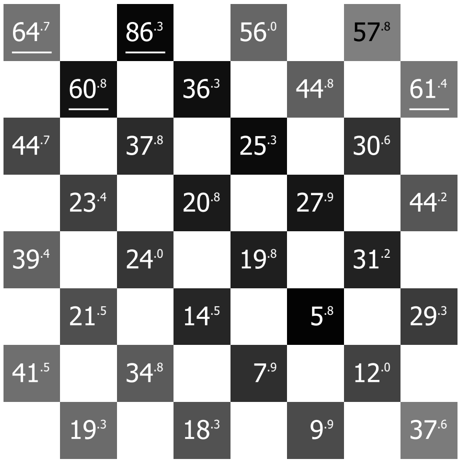
81.1	64.1	57.7	56.1	51.5	59.3	65.5	65.3
47.7	40.9	39.3	37.5	38.3	39.5	47.0	47.4
40.7	45.2	34.6	31.4	33.3	41.9	51.3	57.1
42.1	37.6	34.9	29.3	31.2	41.2	47.8	60.0
45.6	47.2	37.7	30.6	33.6	40.9	49.0	61.4
54.0	45.2	34.7	34.4	32.8	35.0	42.3	61.6
67.6	60.6	51.3	40.3	41.0	40.7	50.7	72.5
30.5	35.0	21.9	17.5	25.8	26.5	46.8	63.0

♜a8



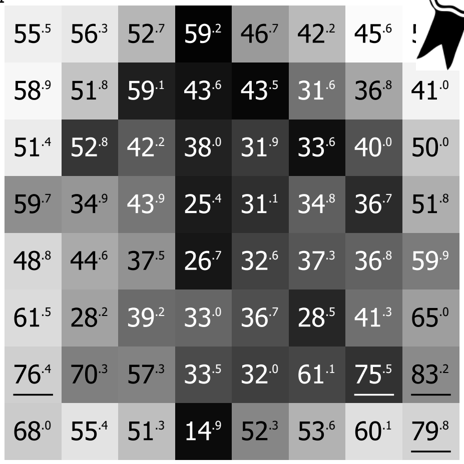
52.4	90.9	55.1	60.5	51.4	62.1	56.8	60.0
52.2	49.4	45.6	42.5	44.4	40.2	51.7	46.8
43.5	49.5	24.5	34.2	30.2	33.3	39.2	42.8
46.1	21.7	26.6	18.3	8.3	27.9	27.9	38.5
32.4	31.7	16.7	9.2	17.6	27.2	33.4	38.4
36.6	11.2	23.0	15.6	21.0	9.9	21.9	36.4
34.3	32.3	35.9	21.2	19.1	29.0	22.1	36.2
30.4	20.6	12.5	18.8	17.5	11.3	20.5	23.6

♞b8



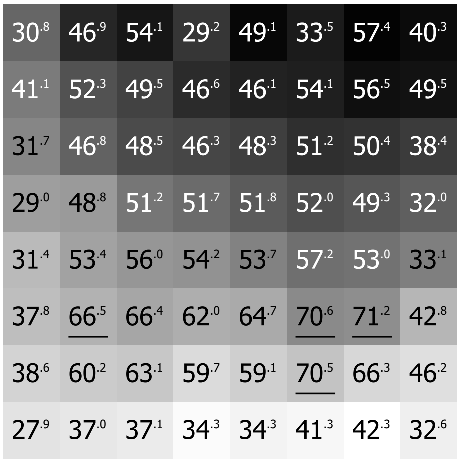
64.7		86.3		56.0		57.8	
	60.8		36.3		44.8		61.4
44.7		37.8		25.3		30.6	
	23.4		20.8		27.9		44.2
39.4		24.0		19.8		31.2	
	21.5		14.5		5.8		29.3
41.5		34.8		7.9		12.0	
	19.3		18.3		9.9		37.6

♝c8



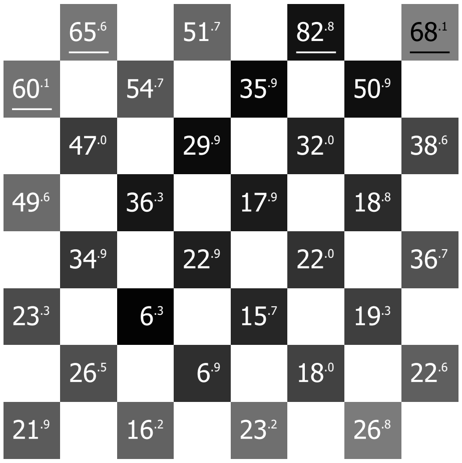
55.5	56.3	52.7	59.2	46.7	42.2	45.6	5
58.9	51.8	59.1	43.6	43.5	31.6	36.8	41.0
51.4	52.8	42.2	38.0	31.9	33.6	40.0	50.0
59.7	34.9	43.9	25.4	31.1	34.8	36.7	51.8
48.8	44.6	37.5	26.7	32.6	37.3	36.8	59.9
61.5	28.2	39.2	33.0	36.7	28.5	41.3	65.0
76.4	70.3	57.3	33.5	32.0	61.1	75.5	83.2
68.0	55.4	51.3	14.9	52.3	53.6	60.1	79.8

♛d8



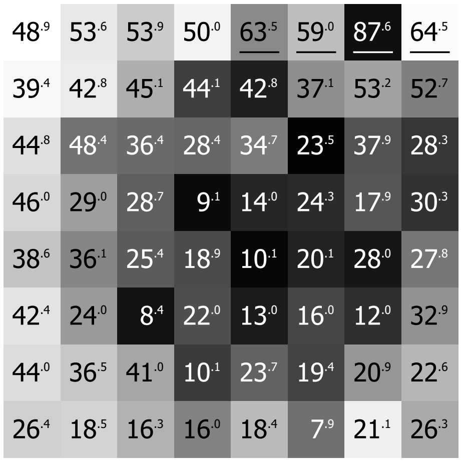
30.8	46.9	54.1	29.2	49.1	33.5	57.4	40.3
41.1	52.3	49.5	46.6	46.1	54.1	56.5	49.5
31.7	46.8	48.5	46.3	48.3	51.2	50.4	38.4
29.0	48.8	51.2	51.7	51.8	52.0	49.3	32.0
31.4	53.4	56.0	54.2	53.7	57.2	53.0	33.1
37.8	66.5	66.4	62.0	64.7	70.6	71.2	42.8
38.6	60.2	63.1	59.7	59.1	70.5	66.3	46.2
27.9	37.0	37.1	34.3	34.3	41.3	42.3	32.6

♚e8



	65.6		51.7		82.8		68.1
60.1		54.7		35.9		50.9	
	47.0		29.9		32.0		38.6
49.6		36.3		17.9		18.8	
	34.9		22.9		22.0		36.7
23.3		6.3		15.7		19.3	
	26.5		6.9		18.0		22.6
21.9		16.2		23.2		26.8	

♝f8



48.9	53.6	53.9	50.0	63.5	59.0	87.6	64.5
39.4	42.8	45.1	44.1	42.8	37.1	53.2	52.7
44.8	48.4	36.4	28.4	34.7	23.5	37.9	28.3
46.0	29.0	28.7	9.1	14.0	24.3	17.9	30.3
38.6	36.1	25.4	18.9	10.1	20.1	28.0	27.8
42.4	24.0	8.4	22.0	13.0	16.0	12.0	32.9
44.0	36.5	41.0	10.1	23.7	19.4	20.9	22.6
26.4	18.5	16.3	16.0	18.4	7.9	21.1	26.3

♞g8

57.8	56.8	52.3	47.9	54.3	73.3	65.1	79.5
43.0	41.3	38.9	36.4	37.5	31.9	45.3	46.2
51.3	46.6	37.6	31.6	32.7	37.4	48.6	49.2
56.8	45.1	38.8	30.7	28.1	34.8	41.8	48.1
63.9	53.1	43.7	33.0	29.7	33.7	44.6	45.8
69.5	54.3	43.9	37.1	29.3	27.3	35.7	50.6
76.1	66.2	55.6	42.1	38.1	40.6	47.6	63.7
64.0	51.4	33.7	21.3	19.5	18.5	38.5	45.6

♜h8

95.6	85.7	74.1	78.4	73.9	64.5	62.0	75.5
82.8	91.2	84.9	85.4	84.6	78.9	82.6	76.1
76.0	52.9	84.7	84.0	82.7	83.2	85.4	84.6
65.2	24.2	38.4	82.2	83.6	82.6	85.8	87.2
55.0	20.2	26.5	75.5	83.2	83.4	87.0	88.8
50.8	18.2	21.0	61.3	85.6	84.9	89.3	91.4
50.9	42.7	44.1	75.9	91.0	91.3	95.0	95.8
48.4	49.7	65.5	83.6	91.5	85.7	95.1	96.5

♟a7

82.1	92.4	83.7	79.4	74.9	63.6	60.4	64.5
84.0	75.4	87.8	86.3	83.8	79.1	82.0	74.1
45.6	73.2	47.4	83.6	82.8	80.8	84.8	82.1
32.8	50.8	35.0	35.0	80.7	83.6	84.8	87.4
19.9	42.3	23.2	22.1	42.9	82.8	86.8	88.5
20.9	33.2	16.7	29.8	39.6	71.2	88.3	91.2
45.2	47.5	40.2	39.8	60.6	77.4	93.5	95.5
51.6	52.2	45.6	54.9	77.7	81.7	94.4	96.4

♟b7

70.5	80.0	89.4	81.9	73.9	64.4	59.4	59.5
77.6	90.0	73.1	86.6	83.4	78.0	81.6	67.3
83.7	52.3	60.4	40.4	82.2	79.9	83.9	81.1
36.4	25.4	44.2	30.9	32.1	82.2	85.3	86.8
29.4	21.9	41.2	6.8	22.9	63.8	86.3	88.5
18.7	15.2	23.9	23.1	19.8	25.9	85.2	90.9
44.7	35.1	43.8	38.5	35.1	38.3	80.9	95.2
62.8	49.4	46.2	44.1	56.9	63.2	92.4	96.1

♟c7

69.0	72.8	76.9	86.7	75.8	69.3	64.5	[illegible]
74.9	87.4	84.3	84.5	83.6	80.6	84.2	[illegible]
82.8	88.1	69.3	59.4	55.1	80.4	85.5	82.5
87.4	37.1	42.2	22.8	20.6	52.3	85.1	87.6
71.9	25.9	12.1	35.5	13.6	21.7	83.9	88.7
55.0	11.9	15.4	26.6	19.9	8.7	40.2	90.1
50.8	32.8	37.9	37.7	35.7	23.8	37.1	83.9
67.3	59.0	41.5	44.7	46.9	44.8	71.2	91.7

♟d7

64.5	68.7	72.1	79.6	87.6	74.4	66.5	62.1
71.0	85.6	82.2	85.8	76.8	81.6	86.3	69.9
81.4	86.8	82.3	44.3	65.5	62.8	86.0	80.9
87.5	87.0	47.4	27.8	30.9	37.3	39.8	86.8
89.7	84.4	19.2	11.3	29.7	16.3	27.8	79.6
92.6	33.7	8.2	21.9	29.2	16.2	19.1	56.1
88.2	10.0	32.0	36.2	40.3	30.8	29.6	42.4
75.1	70.0	46.4	41.6	49.9	43.1	68.5	81.8

♟e7

61.1	63.7	67.1	76.1	83.6	89.5	74.7	65.1
68.4	84.5	80.1	85.3	88.1	72.9	90.4	74.9
80.4	86.3	81.6	83.0	35.0	55.0	52.4	83.1
86.8	87.6	83.6	30.6	22.6	43.6	32.4	32.2
89.9	89.2	51.4	27.6	21.0	40.3	23.1	34.5
92.9	87.8	32.9	25.8	25.9	36.5	17.7	33.8
95.8	87.5	59.6	38.7	40.6	45.1	36.9	38.3
96.9	96.2	81.1	56.6	47.8	53.7	59.1	75.9

♟f7

66.0	64.6	65.3	75.9	78.3	82.6	91.4	76.6
76.2	84.2	81.0	85.0	86.7	87.3	82.7	80.1
81.4	87.2	82.6	83.5	84.0	47.5	70.7	42.4
86.8	86.7	85.6	82.6	35.1	34.9	47.1	32.0
89.3	89.4	84.5	57.8	24.9	24.4	36.5	26.3
93.6	91.6	82.0	50.0	31.4	22.2	35.0	26.7
96.2	96.2	89.3	68.9	42.1	38.3	47.7	41.3
98.1	97.7	94.2	82.9	62.6	51.9	60.6	59.0

♟g7

76.8	66.1	65.3	73.9	75.9	72.4	82.4	95.2
76.7	85.1	80.0	85.2	85.3	84.7	91.8	84.5
83.9	86.4	84.3	82.5	83.8	82.9	56.5	74.1
86.2	87.3	83.2	85.0	81.7	36.7	25.1	62.1
89.5	88.8	84.9	83.9	73.0	24.4	21.6	52.5
93.3	92.1	87.0	86.5	60.5	24.3	16.6	51.9
96.1	96.1	92.9	90.8	74.3	30.4	36.1	51.7
97.7	97.4	94.2	90.9	84.0	60.7	56.5	53.7

♟h7

47.5	49.5	64.2	82.3	91.4	85.1	95.1	96.3
49.8	42.7	44.2	72.5	90.1	91.2	94.7	95.5
49.2	18.7	21.6	54.1	84.5	85.2	89.0	91.6
54.5	18.0	29.1	71.0	82.1	82.8	86.9	88.7
63.6	24.7	38.0	81.1	83.1	81.3	85.9	87.1
74.3	55.0	83.4	83.7	82.2	83.2	85.2	85.2
83.2	91.3	84.4	84.9	84.4	78.5	82.5	76.5
94.8	83.6	72.6	77.5	73.0	65.1	63.4	76.3

♙a2

51.9	52.2	46.0	56.6	78.9	82.0	94.6	96.7
45.3	47.3	41.2	39.5	59.1	76.3	93.8	95.3
21.8	35.6	18.7	30.4	35.4	71.7	88.4	91.7
20.1	40.0	24.0	19.5	43.6	82.3	86.9	88.1
31.6	48.7	34.2	38.9	80.7	82.9	85.1	87.1
45.3	71.4	47.4	83.6	82.7	81.2	85.1	83.7
81.5	74.1	87.0	85.9	83.7	78.9	81.8	75.2
78.1	91.4	83.1	79.2	75.4	64.1	62.8	65.8

♙b2

62.8	49.5	46.7	42.8	54.0	59.2	90.6	95.7
44.8	39.1	44.0	37.7	32.8	30.0	77.2	94.9
17.6	14.3	30.4	25.3	15.4	24.7	85.4	90.8
29.8	18.9	38.7	11.8	25.9	66.3	85.8	88.1
35.7	24.9	40.2	28.8	36.1	80.6	85.0	86.3
82.2	52.7	60.5	42.3	82.3	79.7	84.7	81.7
74.6	89.8	74.0	86.7	84.0	78.2	81.4	68.2
66.9	78.0	88.5	82.6	75.5	65.9	60.7	60.7

♙c2

61.1	57.3	40.5	43.0	44.8	40.3	70.0	[illegible]
50.1	28.3	36.3	36.7	32.9	19.4	32.0	87.7
52.6	13.7	12.5	24.8	14.9	9.5	47.5	90.4
80.9	23.7	11.4	31.6	16.1	22.9	84.6	88.1
87.5	41.0	40.1	24.2	27.0	60.6	84.8	87.8
81.6	88.1	64.6	64.5	55.3	80.6	85.2	83.1
73.3	87.4	82.2	87.3	84.6	79.6	83.3	71.3
66.6	71.7	74.6	86.2	78.7	70.0	65.3	65.1

♙d2

72.4	68.7	40.9	37.8	46.3	36.5	59.3	77.5
91.6	19.3	27.6	32.7	34.2	24.5	19.3	43.6
92.6	54.2	7.7	13.7	23.5	10.6	16.7	65.4
89.4	87.9	25.3	9.5	28.6	16.0	30.1	84.8
87.4	86.5	64.6	30.8	34.5	39.4	48.5	86.8
81.3	87.1	81.5	44.1	69.1	63.6	85.3	82.4
72.1	85.9	81.5	84.4	80.5	81.2	85.6	73.0
64.5	67.9	69.6	78.4	86.1	74.8	69.5	66.7

♙e2

96.7	96.3	78.1	52.2	46.1	51.3	55.9	73.2
96.1	90.4	55.3	33.7	38.8	41.4	33.5	37.1
93.1	89.5	39.5	20.2	19.9	33.3	14.1	33.5
89.9	88.9	57.7	26.5	22.7	33.7	25.1	34.1
86.4	87.8	83.7	33.9	27.3	44.4	32.0	33.7
79.0	86.1	81.3	83.3	40.2	61.5	54.5	84.2
66.3	84.6	80.4	85.7	88.1	79.1	90.1	77.2
59.0	62.4	65.7	75.9	84.6	90.5	78.7	69.3

♙f2

98.1	97.5	93.6	80.4	62.3	50.2	59.3	57.3
96.4	96.3	89.3	65.6	39.4	34.9	47.5	41.3
93.8	91.8	83.7	49.8	27.4	21.7	33.2	26.9
89.2	89.6	84.3	61.9	23.7	21.0	39.4	24.1
86.4	86.7	85.7	82.7	37.8	34.3	47.4	33.5
81.0	86.8	82.2	83.7	84.6	49.1	72.9	41.7
73.9	84.3	80.5	85.5	86.8	87.8	84.8	83.4
64.1	63.1	64.4	75.7	79.2	83.7	92.5	79.7

♙g2

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess

97.5	97.4	93.3	90.2	83.3	58.2	55.0	53.2
96.3	96.3	92.6	90.0	74.3	26.8	34.7	51.7
93.2	92.1	86.8	86.7	55.7	25.0	13.0	54.2
89.4	88.5	84.7	83.2	75.4	22.6	23.2	51.7
86.2	87.5	83.2	85.2	81.6	37.7	25.5	62.6
82.5	86.4	84.6	82.7	84.5	83.5	58.8	76.5
75.1	84.5	80.2	85.5	85.6	85.4	91.3	87.7
74.9	65.2	65.3	74.8	77.7	74.1	84.5	95.7

♙h2

69.8	55.6	51.8	14.6	51.5	54.4	60.9	[illegible]
75.2	69.8	57.6	36.4	36.2	66.8	74.5	[illegible]
60.5	28.4	41.3	33.1	39.8	30.7	46.1	69.2
49.3	43.2	35.5	27.5	31.4	36.2	36.6	61.9
58.7	36.8	43.5	25.4	32.4	35.6	39.0	52.7
53.7	52.0	39.3	43.7	36.7	36.8	43.0	51.7
59.3	49.1	58.6	45.1	44.2	34.4	35.7	45.2
52.8	55.5	50.8	59.3	47.6	41.5	44.9	55.3

♕d1

31.0	35.9	22.1	17.9	25.7	25.9	46.8	60.4
66.0	61.3	52.0	41.1	42.1	41.4	50.3	72.1
52.2	47.3	36.5	35.5	34.3	36.2	42.3	62.1
46.2	47.5	37.0	30.5	33.8	39.0	48.3	60.2
42.3	38.9	33.3	29.6	32.1	40.4	47.7	60.9
42.1	45.7	33.0	35.3	37.1	43.7	53.5	59.9
48.3	38.7	39.0	39.4	40.9	41.0	46.7	49.6
81.7	64.2	55.9	59.7	55.3	60.9	67.0	67.2

♖a1

28.6	38.2	38.1	34.3	33.6	40.2	41.3	31.9
38.3	60.4	64.1	59.7	58.7	69.6	66.2	46.2
37.9	66.8	67.0	62.1	64.7	70.2	71.2	43.3
32.4	54.5	57.1	55.1	54.2	58.4	53.0	34.3
30.1	49.6	52.1	51.8	51.6	52.3	49.3	33.1
32.7	48.1	50.5	47.3	48.9	52.0	51.4	38.9
42.6	53.7	51.2	48.9	48.5	55.5	57.7	51.0
33.0	51.7	59.2	32.2	56.6	36.8	61.5	44.1

♔e1

27.2	21.7	11.4	19.5	19.2	11.4	21.7	24.8
35.8	28.3	39.8	21.0	18.1	29.1	20.2	37.7
35.9	17.8	23.9	19.2	24.0	10.1	22.7	36.0
35.7	29.4	19.9	9.7	18.4	25.3	34.4	37.9
42.5	24.6	27.7	19.5	10.2	28.4	30.0	38.1
44.2	50.7	29.9	37.2	33.3	36.0	42.0	45.1
51.4	45.4	46.1	45.6	46.2	43.8	51.5	51.8
48.5	92.4	53.6	61.7	50.8	62.0	58.0	61.9

♘b1

21.9		15.7		24.3		22.1	
	20.3		5.7		16.8		21.4
19.5		6.1		15.5		20.8	
	31.9		19.9		18.5		37.0
46.2		37.3		17.7		22.6	
	46.1		33.2		33.8		43.6
65.2		59.7		38.6		53.6	
	71.3		54.1		83.2		66.1

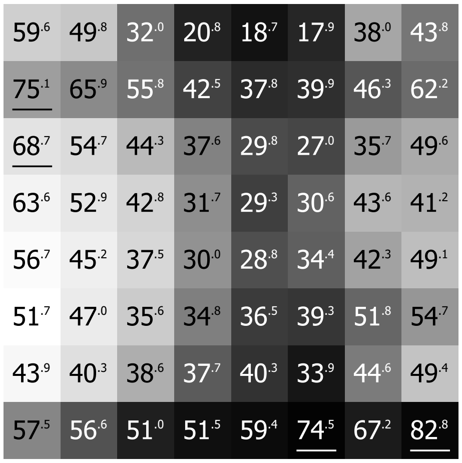
♗f1

	19.0		17.7		9.4		40.2
42.8		38.0		6.5		8.8	
	25.4		18.5		6.6		28.2
44.3		22.8		19.4		30.4	
	26.1		21.3		29.5		41.8
49.1		37.6		30.9		33.3	
	57.9		37.3		47.3		64.9
62.5		86.3		55.2		61.3	

♗c1

25.6	17.5	14.8	17.4	20.2	7.5	21.7	33.8
42.5	32.1	43.7	8.3	20.9	18.7	18.9	23.8
41.1	28.9	6.8	24.6	15.8	16.5	12.3	32.0
40.9	33.0	26.5	19.7	10.1	18.5	28.5	31.9
44.0	30.8	27.5	9.0	15.2	27.9	19.1	28.6
43.9	47.9	37.0	32.6	37.2	25.6	43.9	35.5
38.5	41.4	45.0	43.3	44.1	42.7	52.8	58.1
47.9	51.8	53.7	50.0	60.2	58.3	91.6	63.6

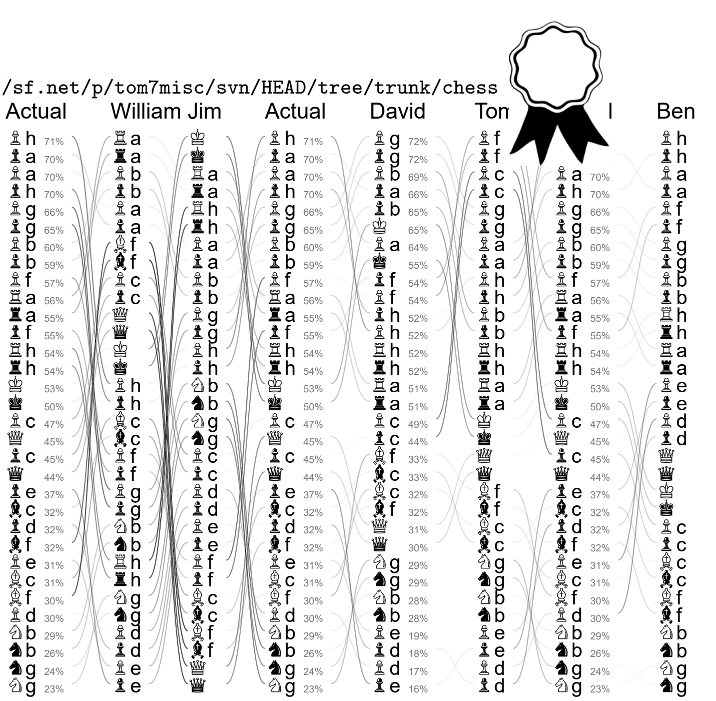
♘g1



59.6	49.8	32.0	20.8	18.7	17.9	38.0	43.8
75.1	65.9	55.8	42.5	37.8	39.9	46.3	62.2
68.7	54.7	44.3	37.6	29.8	27.0	35.7	49.6
63.6	52.9	42.8	31.7	29.3	30.6	43.6	41.2
56.7	45.2	37.5	30.0	28.8	34.4	42.3	49.1
51.7	47.0	35.6	34.8	36.5	39.3	51.8	54.7
43.9	40.3	38.6	37.7	40.3	33.9	44.6	49.4
57.5	56.6	51.0	51.5	59.4	74.5	67.2	82.8

♖h1

Aside from admiring the groovy pictures, the data can also be used as the basis of an algorithm for playing chess. There are several things to try; we could move pieces towards squares where they are more likely to survive than die (keeping pieces alive is good), or just towards squares where they are more likely to be positioned at the end of the game (our moves are more likely to resemble real chess moves because they put pieces in their proper places). Alas, it turns out that these are bad approaches to chess, both because they are boring (most pieces are actually most comfortable in their starting positions) and because they perform badly against even weak opponents [3].



https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess

Figure 2: Piece rankings (from most surviving to most dead); either a human or hypothesis or the actual results across all games. The **actual** column appears multiple times so that each human gets a chance to be adjacent to it; this makes for the easiest visual comparison. Lines connect the same piece in adjacent columns, and are darker if the pairs have more different ranks.

4 Guesses and slices

Like all good scientific research, I explicitly compare the actual results to the hypotheses gathered before the experiment; this is an hygenic and humbling exercise. Figure 2 compares the ranking across all games (slice **Actual**) to the author author (slice **Tom**) and his drinking buddies (slices **Ben**, **Jim**, **David**, **William**). There are a number of different reasonable ways to measure the accuracy of this type of position; a very simple one is the sum of the absolute differences in rank for each piece (e.g. if in one ranking ♔is #3, and in the other #5, then this contributes 2 to the total error). By this metric, Ben has the best prediction (98 error), followed by David (138) and Tom (148). Tom and David had the most similar predictions (116) and Jim and William the most different (312). The expected error between two completely random permutations is about 341, so all of these guesses are significantly better than chance. Note in the actual ranking, many pieces have very similar survival probabilities, and many guesses are ambivalent about groups of pieces. Weighting each rank difference equally is therefore an oversimplification. It would have been better to ask each participant to give probabilities, as David did; this would give us more sensitive error metrics and more opportunities to spend the afternoon making visualizations.

Several drinking buddies gave rationales for their hypotheses (mine appear in Section 1).

Ben does not prefer to use the shift key, a typographic quirk I replicated faithfully here even though it burns my eyes:

> edge pawns almost never played til endgame let alone traded off (♙h, ♙a)
>
> not quite sure where these should go (pb more likely to see play in queenside minority attacks in k-side castle games?) (♙f, ♙g, ♙b)
>
> rook play more likely to be active on q side than on k side (also the classic Nxc7 fork in low rank play), but overall more likely to stay tucked away compared to q (♖h, ♖a)
>
> i think IQP positions are more likely than not saccing e in e4 openings but on the other hand d is often traded off in e4 openings while vice versa is not as true (♙e, ♙d)
>
> q probably involved in many checkmates (low ranked play) or resignations before traded off (high ranked play) (♕)
>
> just randomly guessing k dies in about 1/3 of games, times 1/2 for 2 sides (♔)
>
> this pawn is a super goner (sicilian, QGA, ...) (♙c)
>
> most doomed seem to be the minor pieces as i'd guess at least half of them get traded off on c/f/3/6 or e/d/4/5 in near every game so (♗c, ♗f, ♘b, ♘g)

Jim "barely understands the rules of chess" and "rarely plays." His justifications get "increasingly nonsensical:"

> Most-to-least-survival hero tier list for chess (patch 1.0):
>
> 1.: King — If I estimate that about 2/3rds of all regular pieces are captured in an average game, and the probability of any non-king piece being captured is uniform, then the king is clearly the most likely to

survive. (I'm going to break symmetry here and rank black king less likely to survive than white king.)

2. Both Rooks — Kept in reserve for castling purposes.

3. The A, B, G, and H pawns — maybe people will forget to move them because they are far from the center.

4. Both Knights — They are slippery, but they often get deep into enemy territory quickly.

5. The C,D,E,F pawns — Moved forward to release various more important pieces ⇒ more likely to die.

6. Both Bishops — https://youtu.be/gDnE-5lD7w8

7. The Queen — A high-value target, seems unlikely to survive.

David simply provided a ranking, along with survival probabilities, in typical understated style.

William notes that his guesses are "pretty much off the cuff," but provides some motivated reasoning:

> I figure the king has got to be somewhere near the middle of the pack since he dies in half of games featuring a winner—but with slightly higher-than-even odds of surviving, since some games end in a draw. I'm probably mixing up means and medians here somehow..
>
> I'm gonna assume castling happens more often on the King's side, so let's give Kingside Rook and F, G, and H Pawns a better shot than their fellows on the left. But maybe it should actually be worse, since if they die, it's because they failed to protect the king. Plus, having heard the tip about C Pawn[5] loud and clear, I'm gonna assume that bad boy most often becomes a new Queen, which means he gets more survival points than the real Queen herself.
>
> D and E Pawn are nothing but pawns, and they mostly sacrifice themselves to the cause.
>
> Randomizing within these constraints gives us our starting point. Then the wildcard Bishops and Knights get randomly distributed through what remains to come up with this final answer shown above.

4.1 Slices

The survival probabilities differ depending on the conditions of the game; Figure 3 compares some of those slices. Here the **All** slice is the same as the **Actual** column in Figure 2, and consists of all acceptable games in the database.

The **Titled** slice includes only games where at least one of the players has an official title (Grandmaster, International Master, FIDE Master, etc.[6]). These games have high-quality play, but far fewer samples ("only" 3.4 million). This set exhibits significant variance; for example, the survival rate of ♙c ranges from 38–46%. This is both because of the small sample size and the bucketization by player name; there are few enough titled players that an individual's preference in openings and style of play changes the values of their entire bucket. I caution against reading too much into this column. It seems we can at least conclude that these games tend to be much bloodier (Kings aside, survival rates are lower across the board); top players are less likely to fall for traps early in the game, perhaps more willing to sacrifice material, and more likely to play into endgames where almost every piece is exchanged. If being a chesspiece to the death, you do *not* want to have a Grandmaster playing the game!

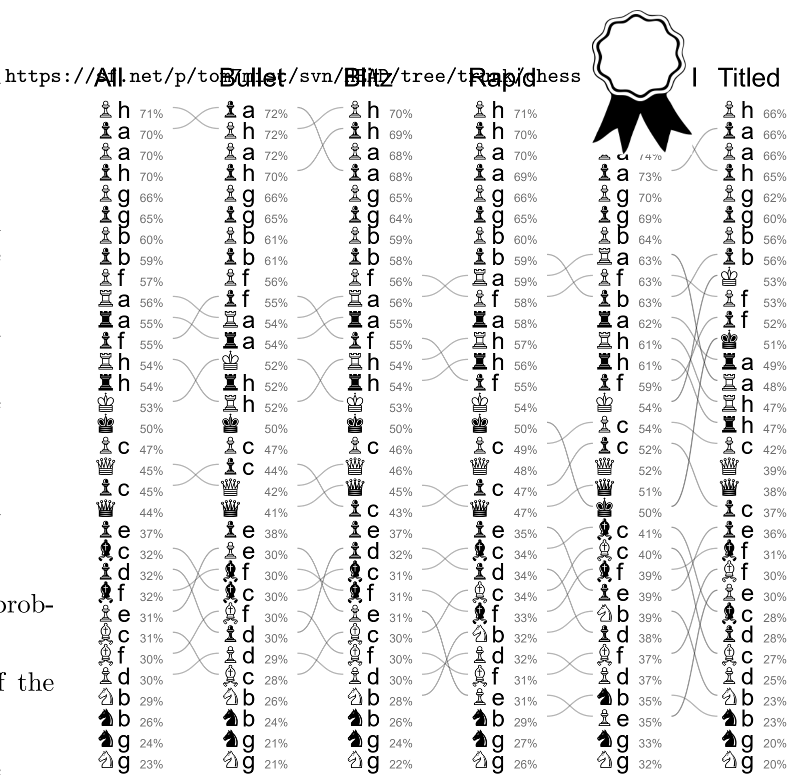


Figure 3: Ranks and survival probabilities for different subsets of games. The same piece in adjacent rows is connected to highlight differences in the ranking, as before. The time formats all exhibit similar ranking with only small perturbations. Games including a titled player (rightmost column) are the most different, although we have far fewer samples in this set, so variance becomes significant.

On the other hand, the other slices all have enough samples that the variance is minimal. These slices, **Bullet** (151,261,707 games), **Blitz** (236,050,938 games), **Rapid** (98,606,558) and **Classical** (13,886,352) are different time control formats. Games on Lichess are played with a starting clock (per side) and an increment added to the clock after each player's move. The game is classified according to the estimated total time: The starting time plus 40× the increment (with the idea that an average game has 40 moves per side); this is the same formula that Lichess uses. A bullet game is when the total time per side is between 30 seconds and 3 minutes; blitz is between 3 and 8 minutes; rapid is between 8 and 25 minutes; classical is any more than this (including untimed games). Each of

[5] I believe the "tip" here was that I described the rest of us (William was the last respondant) as disagreeing most on ♙c. I think William misinterpreted this "tip."

[6] Lichess used to award the LM "Lichess Master" title to notable players on the site; this title is excluded from the sample.

these slices has enough samples that the variance is very low. Here we see that the ranking is rather stable across the range of time formats, which was not what I expected. This should increase our confidence that the results are really inherent to chess, not just the particulars of this data set.

References

[1] FIDE handbook – E.I.01A. Laws of chess, 2017. `www.fide.com/component/handbook`.

[2] Tom Murphy, VII. CVE-2018-90017117. In *A Record of the Proceedings of SIGBOVIK 2019*. ACH, April 2019.

[3] Tom Murphy, VII. Elo World: A framework for benchmarking weak chess algorithms. In *A Record of the Proceedings of SIGBOVIK 2019*. ACH, April 2019.

[4] Tom Murphy, VII, Ben Blum, and Jim McCann. It still seems black has hope in these extremely unfair variants of chess. In *A Record of the Proceedings of SIGBOVIK 2014*, pages 21–25. ACH, April 2014. `sigbovik.org/2014`.

`https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess`

Optimizing *The Sacrifice*

Nico Zevallos

***Abstract*—This work aims to finally fix a glaring issue in Andrei Tarkovsky's final opus *Offret* (The Sacrifice) [1] namely, the inefficiencies of its final six minute shot.**

I. Introduction

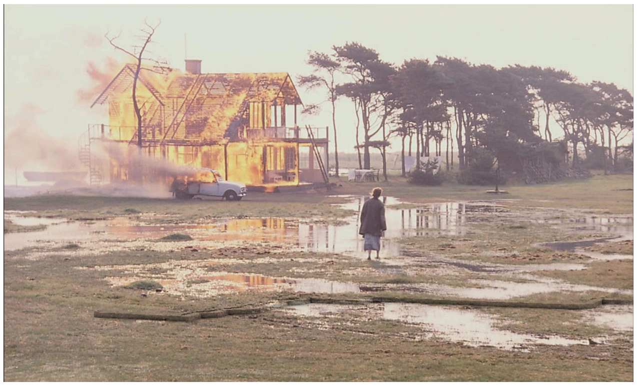


Fig. 1. A frame from the final shot of *Offret*

THERE are perhaps a handful of directors whose oeuvres are as universally venerated as that of Andrei Tarkovsky. Who among us could find fault in the work of a man Ingmar Bergman called "The greatest, the one who invented a new language, true to the nature of film, as it captures life as a reflection, life as a dream [2]." And yet as one watches *Offret*, one error looms. This 1986 masterpiece of Swedish cinema is marred by its famous final sequence. In it, the paterfamilias, Alexander, burns the family's home to the ground in an attempt to honor an agreement with God to undo a nuclear apocalypse. In a single, six minute long take, the family chases him as their house burns to the ground, splashing through and falling into the mud, occasionally collapsing in despair, and finally pushing him onto an ambulance that rolls away into the swamp. The family then watches as their house collapses onto itself in a glorious blaze, all the grandiose posturing of an aging academic conflagrated by a single act of faith.

As one watches this moving scene, one cannot help but think: Couldn't the family have captured and committed its crazed patriarch much more efficiently? The shot itself had to be re-done when a camera jammed during the first attempt, causing the crew to rebuild a perfect replica of the house from scratch to set fire to it a second time. Although nothing can be done at this point about the inefficiencies of the film's production, this paper seeks to optimize the paths the family takes as they rush to confront Alexander such that they may watch their home burn without the additional discomfort of getting their feet wet, as well as complete their task much faster.

II. Recreating the scene

The method for determining optimal, dry paths was as follows. The first step was to track the movement of the camera in the scene. This was done in the 3D modelling software Blender by creating rough models of the static objects in the scene, estimating camera parameters by eye, and manually moving the camera through the scene so that the static objects match up. Based on footage from a documentary about the film [3], the author constrained the movement of the camera to a single degree of freedom of movement along the axis of the dolly that the camera was on as well as two rotational degrees of freedom to account for horizontal and vertical pans. The camera focal lengths tested were 50mm, 70mm, 75mm, 85mm, and 100mm. Focal lengths below 50mm and above 100mm forced the author to make objects such as the house and the cars unreasonably small or large in order to line them up with the footage. After a rough alignment at each of these focal lengths, 75mm was chosen as the ideal focal length because it produced the fewest artifacts and distortions. Following this decision, the footage was meticulously tracked by hand, adding about 1 key frame for every 5 frames of footage and adjusting both the position of the camera and objects in the scene until the entire six minute shot was aligned. Once the shot was aligned, non-static objects representing the characters could also be key-framed to match their position from the camera's point of view, thereby finding their trajectory in the recreated 3D space. The recreated scene, including renderings of the house and cars present in the film as well as Alexander's trajectory through the scene can be seen in Fig. 2.

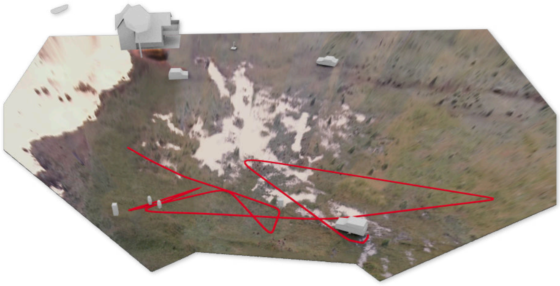


Fig. 2. Recreated scene with Alexander's path overlayed

The second step was to create a cost map for the path planner. The aligned footage was projected onto a plane to recreate the ground plane. This was done by repeatedly finding a frame in which a portion of the ground was visible, and then painting that portion of the ground onto the recreated ground plane similarly to the 'screen space brush' method described by Hanrahan and Haeberli[4]. This reconstructed ground plane had a threshold applied to create a cost map, and unreachable areas such as those inside the house or under cars were manually marked as infinite cost. The resulting cost map can be seen in Fig. 3.

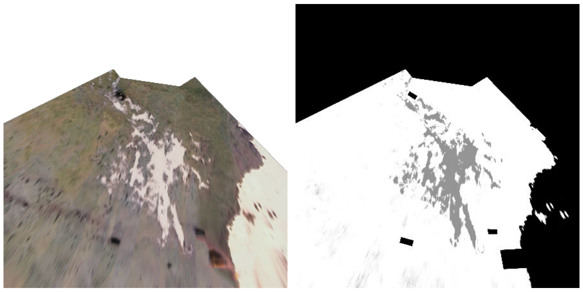


Fig. 3. Texture-painted ground plane on left and generated cost map on right

III. Calculating optimal paths

Once this cost map was created, the optimization could begin. This cost map was used to calculate how much of the path passed through 'water' pixels. This was done using Bresenham's line algorithm [5] to find the pixels covered by the path and finding the ratio of grey (in the water) to white (on dry land) pixels. This cost map was also used in conjunction with the A* planning algorithm [6] as an 8-connected grid in order to find paths that avoided wet areas. A* was chosen as it has the pleasant properties of being extremely fast, complete, and guaranteed to produce optimal paths.

One downside to the use of a grid representation, however, was that the paths were optimal only on the discrete space, where at each pixel the family member only has eight directions they could move. In reality the family could move in an infinite number of directions and weren't restricted to moving from the center of one pixel to another. To mitigate this effect, euclidean distance was used as the heuristic for A* rather than the more traditional diagonal distance used in most 8-connected grid representations. This was chosen because although diagonal distance is admissible in the 8-connected case, it can overestimate distance in the continuous case. In practice what this heuristic means is that paths tend to be closer to what they would be in a continuous space, and less likely to zig-zag and take advantage of cheap diagonal movement. An existing implementation found here was used with some modifications: changing the heuristic to Euclidean distance and changing the cost of diagonal movement to $\sqrt{2}$. Finally, an additional relaxation step was applied in order to produce smoother, shorter paths as proposed by Thorpe and Matthies[7]. An overview of the path-finding is provided in Algorithm 1.

IV. Optimal Path Results

As can clearly be seen in Table I, A* produces perfectly dry paths. Even after relaxation, our method produces paths which are 99% dry. The paths that are taken in the film range from 8-20% wet, leading two characters to fall over face first into the mud. In addition, Table I shows that the paths taken in the film are only marginally shorter than those found by A* and extremely close to those found using Relaxed A*. All of these paths are sub-optimal in terms of distance, which is clearly shown in the comparison to running in a straight line. It is almost as if the characters are going out of their way to get wet. Worse still, neither Marta nor Julia ever reach Alexander, choosing instead to fall to their knees a few meters short and are generally unhelpful for the rest of the sequence. For ease of comparison, this excess distance was added onto their 'Path in Film' lengths. As an additional visualization of the irrational path choice of the characters, Algorithm 1 was repeated for each frame and overlaid over the actual video. The video can be found here.

Algorithm 1: Calculating optimal paths

```
input : pursuer location $\alpha_T$,
        $n$ 2D target locations $\alpha = [\alpha_i, i \in \mathbb{N} : i < n]$
        target location $\alpha_T$
        $m \times n$ grid of costs $G$

output: relaxed A* path $p$

Function Relax($p$, $G$):
    $\nabla G \leftarrow \nabla$ GaussianBlur($G$)
    $gradient \leftarrow [0, 0, ..., 0]$
    for $i \leftarrow 1$ to $maxIterations$ do
        for $i \leftarrow 0$ to Length($p$) do
            $x, y \leftarrow p_i$
            $gradient_i \leftarrow \nabla G_{xy}$
        $optimizeStep \leftarrow gradient + \Delta p$
        $optimizeStep_{start}, optimizeStep_{end} \leftarrow 0$
        if $||optimizeStep|| < 0.1$ then
            break
        $p \leftarrow p - optimizeStep$
    return $path$

Function Main($\alpha$, $\alpha_T$, $G$):
    $p \leftarrow$ AstarSearch($\alpha_i, \alpha_T, G$)
    if Length($path_{new}$) $> 0$ then
        $p \leftarrow$ Relax ($path_{new}$, $G$)
    else
        continue
    return $p$
```

A portion of the sequence was chosen in which the entire family as well as Alexander were all clearly visible, so that we could have an apples to apples comparison of path quality without having to guess the location of off-screen characters. The starting point for each of the characters was set to their location at the first frame of the sequence. Because Alexander stays in approximately the same place for the entirety of this segment, the goal for each path was set to Alexander's median location.

TABLE I
PERCENTAGE OF PATH SPENT IN WATER

Character	A*	Relaxed A*	Path in film	Straight line to goal
Marta	0%	1.2%	12.8%	8.2%
Victor	0%	1.3%	7.7%	21.8%
Julia	0%	1.1%	16.1%	9.7%
Adelaide	0%	1.0%	19.5%	10.0%

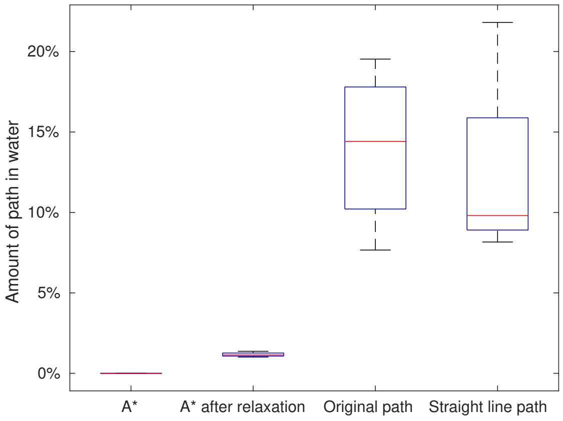


Fig. 4. Box plot showing results from Table I

TABLE II
LENGTHS OF PATHS

Character	A*	Relaxed A*	Path in film	Straight line to goal
Marta	76.7 m	72.8 m	72.0 m	70.7 m
Victor	63.3 m	60.2 m	61.8 m	58.0 m
Julia	84.3 m	80.5 m	79.6 m	77.6 m
Adelaide	82.7 m	78.6 m	79.3 m	76.1 m

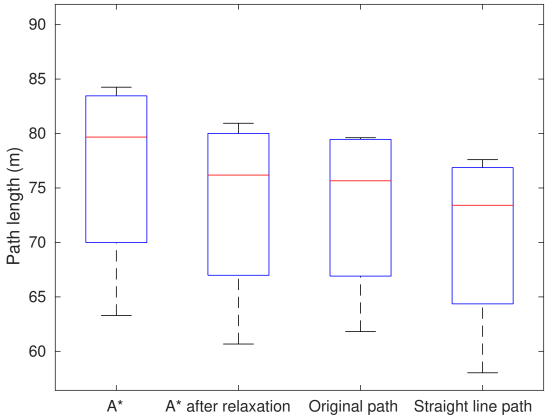


Fig. 5. Box plot showing results from Table II

V. PURSUIT AND EVASION

Although these comparisons are elucidating with regards to pursuer path quality, the author was additionally curious about what would happen if Alexander were actively trying to avoid capture, rather than wandering about in despair. In order to test this behaviour, we formulate the scene as an evasion-pursuit problem. First, a boundary is created which is a convex hull roughly approximating the borders of the cost map, ignoring obstacles.

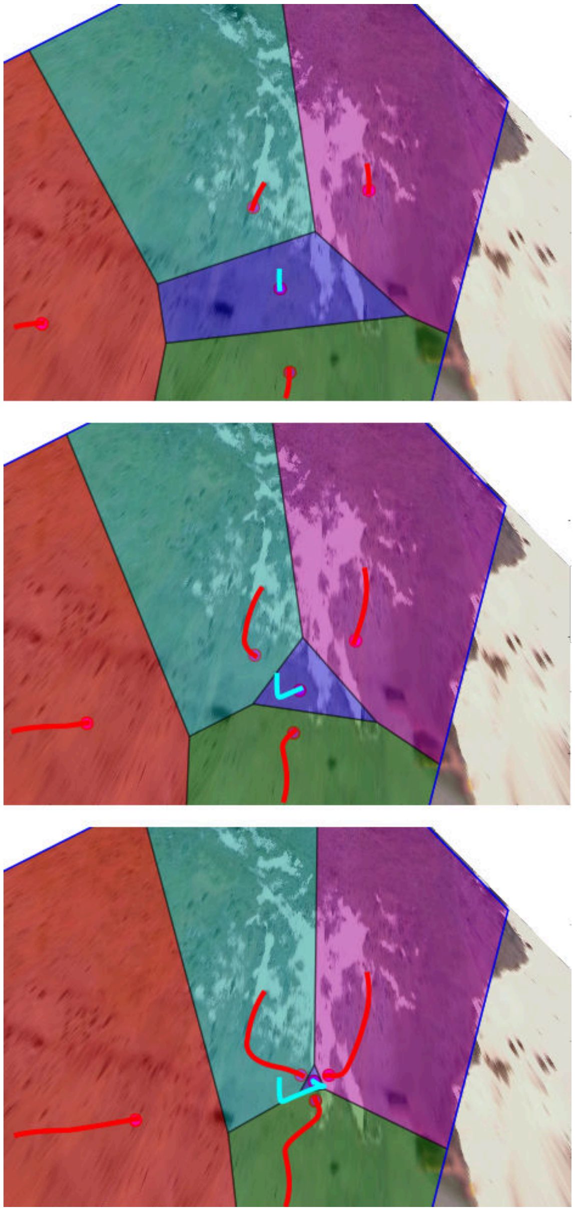


Fig. 6. Result of the pursuit algorithm, with pursuer paths marked in red and evader path marked in cyan.

Alexander employs the fleeing behaviour found in Algorithm 2. This is a very basic fleeing behaviour that avoids pursuers and coastal edges weighted by their distance from Alexander. The pursuers attempt to catch Alexander using Algorithm 3 which was developed by Huang et al.[8]. This algorithm was chosen as it guarantees capture in finite time and can be calculated in real-time. In this algorithm, rather than pursuing the target directly, a Voronoi diagram is constructed. Each pursuer checks if their region of the Voronoi diagram borders the target's region. If it does, they move toward the center of the line defining the border between the two regions as fast as they can. If the pursuers region does not share any border edges with the target region, it simply moves toward the target as fast as it can. This results in the pursuers surrounding the target rather than bunching up during pursuit.

Previous work on this topic only performs simulation experiments on regions with square boundaries[8][9], which makes the calculation of distance to edge as well as the calculation of the bounded Voronoi diagram much simpler. In our case, the calculation was performed by first reflecting all points about every edge in the boundary polygon. The Voronoi diagram was then calculated using the union of the original points and their reflections. These reflections could additionally be used to calculate the distance from any point to all of the boundary edges by taking the differences between a point and each of its reflections and dividing by two.

For the purposes of the experiments, all actors were assumed to run at a human's optimal hunting speed of 3.3 m/s[10].

Algorithm 2: Evasion algorithm

```
in : $n$ pursuer locations $\alpha = [\alpha_i, \forall i \in \mathbb{N} : i < n]$
     target location $\alpha_T$
     convex hull $B$ with $k$ edges $[B_i, \forall i \in \mathbb{N} : i < k]$

out: Direction of evasion

Function CalcPursuerDir($\alpha_T$, $\alpha$):
    $\delta \leftarrow [0, 0]$
    for $i \leftarrow 0$ to Length($\alpha$) do
        $\epsilon \leftarrow \alpha_i - \alpha_T$
        $\delta_i \leftarrow \epsilon / \|\epsilon\|^2$
    return $\delta$

Function CalcEdgeDir($\alpha_T$, $B$):
    $\delta \leftarrow [0, 0]$
    for $i \leftarrow 0$ to Length($B$) do
        $\epsilon \leftarrow$ CalcEdgeDist($B_i$, $\alpha_T$)
        $\delta_i \leftarrow \epsilon / \|\epsilon\|^2$
    return $\delta$

Function Main($\alpha_T$, $pursuers$, $B$):
    $edgeDir \leftarrow$ CalcEdgeDir($\alpha_T$, $B$)
    $pursuerDir \leftarrow$ CalcPursuerDir($\alpha_T$, $\alpha$)
    $\delta \leftarrow \sum edgeDir + \sum pursuerDir$
    return $\delta / \|\delta\| * evaderSpeed$
```

Algorithm 3: Pursuit algorithm

```
in : $n$ pursuer locations $\alpha = [\alpha_i, \forall i \in \mathbb{N} : i < n]$
     target location $\alpha_T$
     convex hull $B$ with $k$ edges $[B_i, \forall i \in \mathbb{N} : i < k]$

out: Direction of pursuit for each pursuer

Function Main($\alpha_T$, $\alpha$, $B$):
    $\alpha_{reflected} \leftarrow$ Reflect([$\alpha_T$, $\alpha$], $B$)
    $regions \leftarrow$ Voronoi($\alpha_{reflected}$)
    for $i \leftarrow 0$ to Length($\alpha$) do
        $edge \leftarrow$ SharedEdge($regions_i$, $regions_T$)
        if $edge \neq \emptyset$ then
            $target \leftarrow (edge_a + edge_b)/2$
            $\delta \leftarrow \alpha_i - target$
            $\delta \leftarrow \delta / \|\delta\|$
        else
            $\delta \leftarrow \alpha_i - \alpha_T$
            $\delta \leftarrow \delta / \|\delta\|$
    return $\delta * pursuerSpeed$
```

VI. Pursuit and Evasion Results

To evaluate the effectiveness of our pursuit strategy, 100 trials were run with random starting positions for all of the actors. For each of these trials, the pursuers were given a maximum of one minute to capture Alexander. Based on observing the original film footage, it was determined that it would take three pursuers to restrain Alexander and thus successfully 'capture' him. Once a pursuer reached Alexander, they would be removed from the Voronoi diagram and travel with him until three pursuers had reached him, at which point the trial was ended and reported successful. If three pursuers had not reached Alexander by the end of the one-minute mark, the trial was reported as a failure.

Alexander was caught in 100% of trials in an average of 16.3 seconds with a standard deviation of 4.5 seconds. The maximum time to capture was 28.0 seconds. In the film, Alexander runs free for a grand total of 5 minutes and 37 seconds before he is forced into the ambulance.

VII. Conclusion

The results of this paper confirm our suspicion that the final sequence of *Offret* contains a grave oversight when it comes to common-sense path planning. Clearly, Tarkovsky and his cast did not do even a cursory literary review. Otherwise, they would certainly have come across A* ([6] was published more than a decade prior to the release of the movie) and path relaxation ([7] was published more than a year before principal shooting began). As our pursuit and evasion results have shown, had the family employed even a simple surrounding technique, they would have caught Alexander an order of magnitude faster. Had the actors and director had the forsight to move in a more coordinated fashion, Tarkovsky may not have had to shoot this sequence twice, and even if he had, he would certainly have saved some money on film. These unfortunate oversights show yet again that the value of meticulous academic research trumps the brash, emotional decisions of even the greatest artists. As roboticists, scientists, and people of Reason, we continue to wonder: "Will they ever learn?"

References

[1] Andrei Tarkovsky. *Offret*. Svenska Filminstitutet (SFI), 1986

[2] Paul Coates. *Film at the Intersection of High and Mass Culture*. Cambridge University Press. pp. 157-158, 1994.

[3] Michal Leszczylowski *Regi Andrej Tarkovskij*. Svenska Filminstitutet (SFI), 1988

[4] Pat Hanrahan, Paul Haeberli. Direct WYSIWYG Painting and Texturing on 3D Shapes. *Computer Graphics*, vol. 24, no. 4, pp. 215-223 August 1990.

[5] Jack Bresenham. Incremental Line Compaction. *The Computer Journal*, vol. 25, no. 1, pp.116-120, February 1982

[6] Peter E. Hart, Nils J. Nilsson, Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Intelligence/sigart Bulletin - SIGART* vol. 37. pp. 28-29, 1972.

[7] Charles E. Thorpe, L.H. Matthies. Path relaxation: Path planning for a mobile robot. *Proc. AAAI*, pp. 318-321, 1984.

[8] Haomiao Huang, Wei Zhang, Jerry Ding, Duan M. Stipanovi, Claire J. Tomlin. Guaranteed Decentralized Pursuit-Evasion in the Plane with Multiple Pursuers. *Proceedings of the IEEE Conference on Decision and Control*, 2011.

[9] Zhengyuan Zhou, Wei Zhang, Jerry Ding, Haomiao Huang, Duan M. Stipanovi, Claire J. Tomlin. Cooperative pursuit with Voronoi partitions. *Automatica* vol. 72, pp. 64-72, 2016.

[10] Karen L.Steudel-Numbers, Cara M.Wall-Scheffler. Optimal running speed and the evolution of hominin hunting strategies. *Journal of Human Evolution*. vol. 56, no. 4, pp. 355-360, April 2009.

https://github.com/amazonlinux/rust-bundled-packaging/tree/b9c50d16b6d517c4e7483c6842b6f3cc77969b9d

Abusing the RPM Package Manager to Compile Software

iliana destroyer of worlds[*]
Linux Witch
Amazon Web Services
iweller@amazon.com

ABSTRACT

The RPM Package Manager (RPM) is a popular Linux utility designed to destroy your computer. We extend its destructive capabilities to include dynamically building the list of upstream code sources based on the list of upstream code sources.

ACH Reference Format:
iliana destroyer of worlds. 2019. Abusing the RPM Package Manager to Compile Software. In *Proceedings of SIGBOVIK 2019, Pittsburgh, PA, USA, April 1, 2019 (SIGBOVIK '19)*, 3 pages.

1 INTRODUCTION

The RPM Package Manager (RPM) [4] is an extremely dangerous Linux program that should be avoided at all costs. Effects of use include destruction of data, privileged arbitrary code execution, and in the worst case, installation of a Linux operating system. Many Linux distributions use RPM as their primary package manager; even more don't.

There are two main modes for using RPM: compiling software, usually with the *rpmbuild* command, and installing software, usually with the *rpm -i --force --nodeps*[1] command.

Modern programming languages, such as Perl, have their own package management ecosystems. Within these ecosystems, seemingly innocuous software can depend on multiple incompatible versions of the same dependencies. This is, of course, an impossible issue to solve. It is core to every packager's instinct to add all existing development libraries to a Linux distribution, regardless of how useful those libraries are to an end user, and to build all tools against distribution-packaged libraries.

Firecracker, an advanced chroot developed by AWS, is one such package with many Rust dependencies. Nevertheless, we added Firecracker to Amazon Linux, a Linux distribution maintained by a book store, without building hundreds of useless RPMs. The solution we developed can, but should not, be applied to other programming language ecosystems with intractable dependency closures.

Most importantly, the solution is a complete hack that shouldn't work and should have never been written.

2 RPM: THE GOOD PARTS

Recipes for RPM are called *spec files* (a sample is provided in Listing 1). They begin with general metadata about a package, called the *preamble*, such as the name, mailing address, Social Security number, source tarball, and any patches. This is followed by the *scriptlet* sections, which are Bash scripts; *%prep* unpacks and sews patches onto the source code, *%build* builds it, and *%install* installs it into a build root. Finally, *%files* lists the files installed from the build root, and *%changelog* lists who broke your business-critical software. [1]

Listing 1: A spec file

```
Name: robotfindskitten
Version: 2.7182818.701
Release: 4%{?dist}
Summary: A zen simulation
Source0: http://robotfindskitten.org/.../%{name}-%{version}.tar.gz
Patch0: nki-makefile.patch

BuildRequires: ncurses-devel

%prep
%autosetup -p1

%build
%configure
%make_build

%install
%make_install

%files
%{_bindir}/%{name}

%changelog
* Sun Dec 10 2017 iliana weller <ilianaw@buttslol.net>
- Update to 2.7182818.701 (#1297151)
```

Note the complete lack of any common shell commands which one normally finds in Bash scripts. They're all hidden away behind *macros* to take away any artistic expression that software packaging once had, ensuring that all code is built with exactly the same *configure* and *make* options. [3] To ensure consistency in a distribution, this is coupled with a stringent and continuous review process... wait, spec files often get reviewed once and never again? Oh.

When a user feeds the spec file to *rpmbuild*, it most likely fails spectacularly. Sometimes it doesn't, though, and the user gets RPMs as output. RPMs are kind of like a candy-coated tarball; a hard, metadata-rich exterior breaks open to reveal a gooey *cpio* filling.

2.1 Thousands of Packages You Can't Use

A common rule in Linux distributions is "one project, one source package" [3]. This means that the TEX Live distribution in Fedora is represented by a single 220,000-line spec file generated by a script [5], while each of the 780,000 dependencies available on npm must have a separate spec file.

[*] Uppercasing of name is punishable by intergalactic law. Views expressed do not necessarily reflect Amazon's.

[1] -i --dont --force --nodeps and neither should you.

https://github.com/amazonlinux/rust-bundled-packaging/tree/b9c50d16b6d517c4e7483c6842b6f3cc77969b9d

Dedicated Python scripts, assisted by obedient packagers, take this rule to its extreme. In Fedora 27 there were 1,129 package names prefixed with *nodejs*, 465 package names prefixed with *golang*, and 132 package names[2] prefixed with *rust*. [5] The scripts can recursively find dependencies, commit spec files, and start builds.

Fedora is not alone; Debian has over 450 Rust source packages; each one builds several binary packages for different feature flags.

These packages commonly install source code to distribution-defined file paths that are ignored by the tools that know how to compile this code. They are dead weight in repository metadata, and adding too many Rust packages incurs the risk of package repository oxidization.

2.2 One Version Ought To Be Enough For Anybody[3]

fd[4] is a Rust utility for finding files. We inspected the Cargo.lock file for fd version 7.3.0[5], which lists its full dependency closure. In order to compile fd, we need two versions of *rand_core* (0.3.x and 0.4.x), as well as two versions of *winapi* (0.2.x and 0.3.x).

In the usual Linux distribution dependency model, this means you need to have two distinct versions of the *winapi* package installed simultaneously, which RPM front-ends[6] did not support until recently, and you need the ability to specify a permitted version range (e.g. $0.3.0 \leq v < 0.4.0$), which RPM did not support until late 2017 (version $4.14 \leq v < \infty$).

This leaves longer-support distributions no option for building Rust code, which is becoming required for common staples of Linux distributions (most notably Firefox, librsvg, and vape[7]).

Well. There is *an* option, but we don't like it.

3 (DEFUN GETSOURCES () (GETSOURCES))

RPM macros are very powerful, incredibly dangerous, and an excellent example of write-only code [2]. The *%autosetup* macro in Listing 1 is responsible for unpacking and patching code. Its definition in Fedora 29 is reproduced in Listing 2.

RPM macros can perform option parsing and introspection of sources and patches defined in the spec file. They can also run arbitrary Bash or Lua code. All of this can be mixed together, and nothing is off-limits; macros can write out entire scriptlet sections, preamble fields, and even Harry Potter slash fiction.

What if we could abuse macros and kill off those thousands of unusable packages? Our macros would need to:

- Introspect a source file for all its Rust dependencies, and add *Source#* lines to the preamble
- Fake a Cargo registry containing all these dependencies
- Build and install the target binary

We built these macros, which contain an unhealthy mixture of JSON parsing, AWK, and Lua. Not only did we build them, we distributed them in an actual Linux distribution, we built Firecracker using them, and we made the code rebuildable by users via retrieving the source RPM and using *rpmbuild --rebuild*.

Listing 2: The %autosetup macro

```
%autosetup(a:b:cDn:TvNS:p:)\
%setup %{-a} %{-b} %{-c} %{-D} %{-n} %{-T} %{!-v:-q}\
%{-S:%global __scm %{-S*}}\
%{expand:%__scm_setup_%{__scm} %{!-v:-q}}\
%{!-N:%autopatch %{-v} %{-p:-p%{-p*}}}
```

For your safety (and because TEX is difficult), the macros[8] are only partially reproduced in Listing 3. At your own risk, they are available on GitHub[9] and in the Amazon Linux source RPMs.

For a packager to download all the required source files, they must run the spec file through a spec file parser (such as *rpmspec -P*), then use a source download tool (such as *spectool -g*).

3.1 Okay, So Not Everything's Perfect

For some dependencies to build correctly, packagers must include the necessary *BuildRequires* and *Patch#* lines to allow them to build. In this system, these must be duplicated across every package that shares the same dependency. This can be resolved by arbitrarily limiting the number of packages that use this solution, perhaps by utilizing the tried-and-true mechanism of "laziness".

4 FURTHER APPLICATIONS

The general approach described here can be used for any programming language with lock files describing their dependency enclosures and with standard URLs to fetch source code, such as Node.js or what people wish Go would be.

We *really* shouldn't though.

5 FURTHER TANGENTIALLY RELATED RESEARCH TOPICS

Calculate the total bandwidth used to transfer the *%changelog* section of the TEX Live spec file, duplicated in each of its nearly 6,000 binary RPMs [5], across Fedora and its derivative distributions.

Something something Nix Docker Flatpak AppImage snaps.

ACKNOWLEDGMENTS

The author acknowledges Amazon for inexplicably employing her.

Thanks to Natasha Jarus, Samuel Karp, Euan Kemp, Tom Kirchner, and Colleen Quine for their reviews.

REFERENCES

[1] Edward C. Bailey. 1997. *Maximum RPM.* Sams Publishing. http://ftp.rpm.org/max-rpm/
[2] Wikipedia contributors. 2019. Write-only language. *Wikipedia* (2019). https://en.wikipedia.org/w/index.php?title=Write-only_language&oldid=880031540
[3] Tom 'spot' Callaway et al. [n. d.]. Fedora Packaging Guidelines. https://docs.fedoraproject.org/en-US/packaging-guidelines/
[4] Erik Troan, Marc Ewing, et al. 1995. RPM Package Manager. http://rpm.org
[5] Will Woods. 2018. Unpacking RPM: package names. https://weldr.io/Unpacking-RPM-names/

[2] 132 is a lot for the first Fedora release where Rust packages were permitted at all; the current count is over 500.

[3] With apologies to Bill Gates, noted proponent of Linux.

[4] fd stands for Finger Donuts.

[5] https://github.com/sharkdp/fd/blob/7f58e8f7064346ffe569563b657fe92f24830e6a/Cargo.lock

[6] Common RPM front-ends include Zypper, yum, and DNF (which stands for Do Not Fuck).

[7] https://github.com/JoshuaRLi/vape

[8] Object Class: Euclid

[9] https://github.com/awslabs/rust-bundled-packaging/tree/b9c50d16b6d517c4e7483c6842b6f3cc77969b9d

https://github.com/amazonlinux/rust-bundled-packaging/tree/b9c50d16b6d517c4e7483c6842b6f3cc77969b9d

Listing 3: RPM macros for bundling Rust dependencies

```
# SPDX-License-Identifier: MIT

# Copyright (c) 2017 Igor Gnatenko
# Copyright 2018 Amazon.com, Inc. or its affiliates.

%__cargo %{_bindir}/cargo
%__cargo_common_opts %{?_smp_mflags}

%_cargometadir %{_datadir}/cargo-metadata

%cargo_prep (\
set -eu \
%{__mkdir} -p .registry \
REGISTRY=”$(realpath .registry)” \
%{__mkdir} -p .cargo \
cat > .cargo/config << EOF \
[build]\
rustc = ”%{__rustc}”\
rustdoc = ”%{__rustdoc}”\
rustflags = %{__global_rustflags_toml}\
\
[term]\
verbose = true\
\
[source]\
\
[source.local-registry]\
directory = ”$REGISTRY”\
\
[source.crates-io]\
registry = ”https://crates.io”\
replace-with = ”local-registry”\
EOF\
do_cargo_build_registry() { \
%__cargo_build_registry \
} \
do_cargo_build_registry $REGISTRY \
)

%cargo_build %{__cargo} build %{__cargo_common_opts} --release %{?cargo_args}

%cargo_test %{__cargo} test %{__cargo_common_opts} --release --no-fail-fast %{?cargo_args}

%cargo_install \
%{__cargo} install %{__cargo_common_opts} --path . --root %{buildroot}%{_prefix} %{?cargo_args} \
%{__rm} %{buildroot}%{_prefix}/.crates.toml \
%{__mkdir_p} %{buildroot}%{_cargometadir} \
%{__cargo} metadata --format-version 1 %{?cargo_args} > %{buildroot}%{_cargometadir}/%{name}.json

%__cargo_crate_source_url() https://crates.io/api/v1/crates/%1/%2/download#/%1-%2.crate
%__cargo_crate_source_urls grep ’ˆ”checksum’ | awk ’{ print ”Source” NR+9999 ”: %%{__cargo_crate_source_url ” $2 ” ” $3 ”}” } END { print ”%%global __cargo_first_crate 10

%__cargo_build_registry %{lua:for i = rpm.expand(”%__cargo_first_crate”),rpm.expand(”%__cargo_last_crate”) do \
   uncompress = rpm.expand(”%{uncompress:%{S:” .. i .. ”}}”) \
   print(uncompress .. ” | tar -x -C $1\\n”) \
   template = ’{”files”:{},”package”:”%s”}’ \
   print(”printf ’” .. template .. ”’ $(sha256sum ” .. rpm.expand(”%{S:” .. i .. ”}”) .. ” | awk ’{ print $1 }’) > $1/$(” .. uncompress .. ” | tar -t | head -n 1 | cut -d / -f 1)/.cargo-chec
end}

%cargo_bundle_crates(n:t:l:) \
%{-t:%{-l:%{error:cargo_bundle_crates: Can’t specify both -t and -l}}} \
%{!-t:%{!-l:%{error:cargo_bundle_crates: Must specify one of -t or -l}}} \
%{-t:%{expand:%(%{uncompress:%{S:%{-t*}}} | tar -xO %{-n:%{-n*}}%{!-n:%{name}-%{version}}/Cargo.lock | %__cargo_crate_source_urls)}} \
```

Security and privacy

Common Vulnerabilities and Exposures

CVE-2018-90017117

Also known as: #KingMe attack

Published 2018-12-03

Partial embargo until 2019-04-01

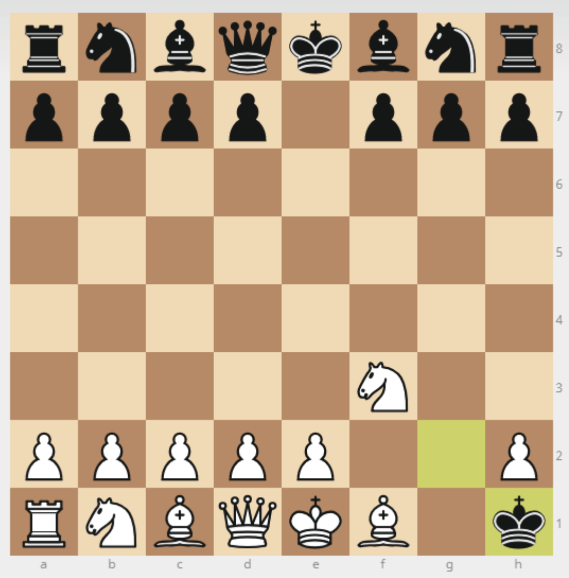
Reported by t0m7 of East Coast Hacking Organization

Description

An input validation error in the move parser allows remote privilege escalation.

Background

The popular internet chess site lichess.org allows for the import of PGN files, a standard text-based interchange format for giving the sequence of moves in a game. Moves look like “e4” (move a pawn to the e4 square) or “Qxd3” (queen captures on d3) or “Rcc8” (the rook on the C file moves to c8). When a pawn moves into the last or first rank, it usually promotes to queen, but may legally promote to a bishop, knight, or rook at the player’s option. This preference is specified using the notation g8=B (or N for knight, R for rook, or Q for queen to optionally be explicit). lichess.org does not properly implement this syntax, and allows a move like g8=K, which is not legal chess.



Screenshot. After 5. ... gxh1=K ?!, black promotes their g pawn to a second king.

Impact

The pawn is promoted to a king. This is a privilege escalation vulnerability, because the king has privileges that the pawn does not have, such as the privilege to be checkmated.

Scope

The issue is only confirmed during PGN import (e.g. in “analysis board”). In live games, it is possible to use keyboard entry of moves in PGN notation, but =K is ignored. It is possible that these moves are only rejected in the frontend and would be allowed by the underlying chess engine (if made directly through the API, for example). After a second king is introduced, the game appears to be quite broken; some parts of the interface behave as though the game is a draw and no further moves are allowed, but the computer continues to suggest lines in the background. When evaluating this vulnerability in other systems, note that the king has not yet moved, and so could erroneously be considered eligible to castle (e.g. with the h8 rook), a potential 0-0-day.

Example exploit

1. f4 e5 2. Nf3 exf4 3. g4 fxg3 4. Ng1 g2 5. Nf3 gxh1=K

Classification - Office Use Only

```
CVSS v3.0 Severity and Metrics:
Base Score: 7.7 HIGH
Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N (V3 legend)
Impact Score: 7.9
Exploitability Score: 8.9
```

Their system is live at: `http://orchhit.com`

Orchhit: User-Oblivious Social Networking

JIM MCCANN[*], TCHOW llc

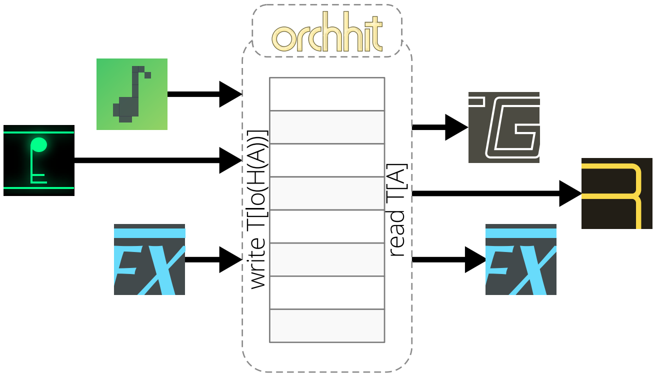


Fig. 1. Our user-oblivious social network, Orchhit, allows sharing of orchestral-hit-sound status updates in a user-oblivious way, thanks to a preimage-writable table construction.

In this paper, we describe the unique infrastructure used by our new user-oblivious social network, Orchhit (Figure 1). This infrastructure uses a constant-sized storage to support basic status sharing for an unlimited number of users; allows instant client-side account creation and deletion; and is immune to server-side snooping. Since we are magnanimous co-founders, we reveal our infrastructural secrets in this tell-all publication.

CCS Concepts: • **Security and privacy** → *Privacy protections*; Hash functions and message authentication codes; • **Networks** → *Social*.

Additional Key Words and Phrases: privacy, failed social networks, overly casual writing, isn't it unfortunate that cryptozoology has nothing to do with cryptosystems?

1 INTRODUCTION

Modern social network web *page* (site) companies face three major problems: acquiring new users, providing server resources to support existing users, and repressing their own inescapable desire to sell users' private information. We present a novel social network architecture – user-oblivious social networking – that uses cryptographic primitives to mitigate all three of these problems.

Particularly, our user-oblivious social network backend provides a status sharing platform and the following guarantees:

(1) Instant client-side account creation and deletion without server contact (thus, no way for operators to determine the number of accounts).
(2) Friends lists are never stored on the server in a way that can be read by operators (thus, no way for operators to determine the social network of accounts).
(3) Uses a constant amount of server storage.
(4) Provides no way to distinguish status updates from random (or user account) data, unless status updates are improperly designed.

2 BACKGROUND

Learning from the mistakes of others would only slow us down.

3 METHOD

Our user-oblivious social network construction is built on a cryptographic hash function $H(X) : \mathbb{Z}_2^* \to \mathbb{Z}_2^{2B}$ which maps arbitrary-length bit-strings to fixed-length bit strings of length $2B$ in a way that is hard to invert and does not induce any correlations between output bits (along with some other important properties that we can't be bothered to look up right now).

3.1 Server

The social network server is responsible for persistent storage of a 2^B-entry table of B-bit storage locations, T. This table is initialized with random bits.

The server provides two interfaces, `read` and `write`, to the client to manipulate the table. Read allows a client to retrieve the table value at a given B-bit address:

$$\begin{aligned} &\texttt{read}(A : \mathbb{Z}_2^B) : \\ &\qquad \textbf{return } T[A] \end{aligned} \tag{1}$$

Write allows a client to set the table value at any address for which it knows a B-bit hash preimage:

$$\begin{aligned} &\texttt{write}(P : \mathbb{Z}_2^B, V : \mathbb{Z}_2^B) : \\ &\qquad T[\texttt{lo}(H(P))] \leftarrow V \end{aligned} \tag{2}$$

Where $\texttt{lo}(B) : \mathbb{Z}_2^{2B} \to \mathbb{Z}_2^B$ is the function that returns the low B bits of a $2B$-bit value.

These two primitives – `read` and `write` – are the only things that the server implements in Orchhit – the remainder of the social network relies on client operations exclusively.

[*] ix@tchow.com

Their system is live at: http://orchhit.com

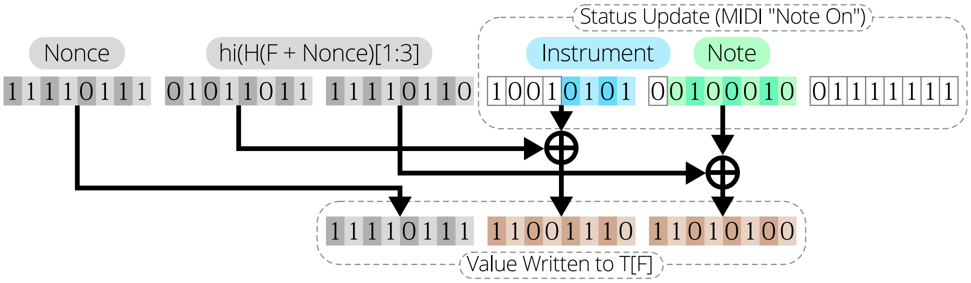


Fig. 2. The status format in Orchhit is a MIDI note-on message whitened by a nonce.

3.2 Client

Notice that the server operations described above allow secure publish-subscribe interaction between clients – one client can pick a value P to write status updates to and publish value $\texttt{lo}(H(P))$ to enable other clients to read these updates. This interaction forms the basis of our platform's social networking operations.

Login / account creation. To start using our social network, a client picks a passphrase, P; computes a secret key, $K \equiv H(P)$, by hashing the passphrase; and derives a follow key, $F \equiv \texttt{lo}(H(\texttt{lo}(K)))$, by hashing the low bits of the secret key.

Notice that this process does not require any server communication or storage. Note, also, that the high bits of the secret key will never be sent to the server.

Account deletion. Account deletion can be performed by forgetting the passphrase.

Status updates. To publish a status update, the client writes the update using its secret key:

$$\begin{aligned}&\texttt{publish}(U):\\&\qquad\texttt{write}(\texttt{lo}(K), U)\end{aligned} \tag{3}$$

Friends list. To store or retrieve the ith element of its friends list, the client uses addresses, P_i, derived from its secret key, obscuring the contents with a value, X_i, derived from the extra-secret upper bits of its secret key:

$$X_i \equiv \texttt{lo}(H(\texttt{hi}(K) + i))$$
$$P_i \equiv \texttt{lo}(H(\texttt{lo}(K) + i))$$

$$\begin{aligned}&\texttt{getFriendAddress}(i):\\&\qquad\textbf{return } \texttt{read}(\texttt{lo}(H(P_i))) \oplus X_i\end{aligned} \tag{4}$$

$$\begin{aligned}&\texttt{setFriendAddress}(i, A):\\&\qquad\texttt{write}(P_i, A \oplus X_i)\end{aligned} \tag{5}$$

Notice that no provision is made for variable-length friends lists. In our implementation, every user has exactly four friends.

4 PROPERTIES

The above construction is simple but provides some very useful properties which make it hard to deduce anything about the social network by inspecting a snapshot of its storage.

User Account Obliviousness. Since all of a user's account data (their friends list) is xored with a passphrase-dependent stream, it is impossible to distinguish user accounts from empty (initialized-to-random) storage.

Constant Storage. Rather than consuming ever-increasing amounts of storage, the server's table remains fixed-size for the life of the network. As the number of users grows, the user experience will gracefully degrade as friends list entries and statuses get over-written by hash collisions. As the designers of the internet understand, this sort of "gentle failure" is much preferable to a hard failure, and may even lead to self-regulation, as users will quit using the (apparently flaky) system.

Status Update Obliviousness. As long as status updates are IID[1], and fill the entire encoding space, status updates stored in the table are also indistinguishable from random bits.

At present, this is an implementation detail that the client needs to manage[2].

5 IMPLEMENTATION

Our user-oblivious social network, Orchhit, is live at http://orchhit.com. It provides users the ability to push orchestral hit sounds to their followers and to follow up to four friends. Status messages are, therefore, MIDI note-on messages, which are whitened using a nonce and a hash value in a way that is somewhat flawed, Figure 2, though red-teaming this is left as an exercise to the reader .

For our implementation, we chose $B = 24$ as a reasonable compromise between usability and security[3]. As a hash function our implementation uses SHA-1 (truncated to 48 bits) because implementations are readily available and because we clearly lack cryptographic acumen.

As a bandwidth saving measure, the read call implemented by our server provides the option to defer return until the value is different from a provided value. This technique, termed *long polling*,

[1] "Locally owned and responsibly sourced."
[2] We *certainly* didn't get this wrong in our client.
[3] Is 2^{24} a sufficiently large number? Of course it is! It's higher than most people can count even if they use binary and all their fingers and toes.

Their system is live at: http://orchhit.com

avoids some bandwidth costs and *probably* doesn't open the system to any weird timing attacks.

6 FUTURE WORK

The astute among you may have realized by now that selecting a secure value for B may be impossible, especially as compute efficiency seems to be growing much faster than storage efficiency. One solution might be to use a pearl-diver construction to provide proof-of-work along with read requests[4].

For some social network designs, allowing only four friends may seem limiting. However, this limit can easily be overcome by realizing that your definition of "friend" is not sufficiently narrow (or by keeping multiple accounts open).

Unfortunately, our system is vulnerable to several side-channel information disclosure attacks. Anyone able to observe the network traffic to and from the server – e.g. the social network operator themselves – can estimate the type of individual storage locations by observing the read and write behavior over time. This information may be used to determine the number of users and – potentially – the contents of their status updates. In our present network, this hazard is largely mitigated by the fact that status updates are just orchestral hit sounds so, like, chill out folks.

7 CONCLUSION

In this paper, we described a construction for a user-oblivious social network based on a preimage-writable table[5].

Readers are encouraged to try our social network at http://orchhit.com. Philosophically speaking, you either already have, or can never truly have, an account.

We hope that our work points the way forward for a bold new set of web services that use cryptographic primitives to make scaling easy and monetization nearly impossible.

ACKNOWLEDGEMENTS

Thanks to Brian Saghy for extensive conceptual development assistance, domain administration, and being too humble a co-founder to want authorial credit.

[4] This may enable blockchain-backed automated fulfillment services to support big-data-centric AI-first omnichannel retailtainment; enhanced, of course, by a sustainable and authentic brand story. Yes, just put the venture capital money over there.

[5] Which we have actively avoided looking up prior work on, preferring to treasure the illusion of novelty.

Machine learning

11 Color- and piece-blind chess

Dr. Tom Murphy VII Ph.D.

Keywords: chess, handicaps, man-in-the-middle attacks, neural networks

12 Dimensionality-reducing encoding for classification of Pythagorean engendered numbers

Rany Tith and Oscar I. Hernandez

Keywords: number theory, machine learning, encoding, information theory, classification, gender theory, integers, globalization, computation, mathematics

13 emojizip: A text compression system based on pictogram-kiloword equivalence

William Gunther and Brian Kell

Keywords: data compression, TensorFlow, laughing crying emoji

14 Meta-meta-learning for neural architecture search through arXiv Descent

Antreas Antoniou *et al.*

Keywords: meta, meta-meta, deep, NAS

15 Towards automatic low hanging fruit identification for the steering of ML research

Nick Frosst and Aidan Gomez

Keywords: machine learning, detection, segmentation, orientation, apple, akee, apricot, avocado, banana, bilberry, blackberry, blackcurrant, black sapote, blueberry, boysenberry, Buddha's hand (fingered citron), crab apples, currant, cherry, cherimoya (custard apple), chico fruit, cloudberry, coconut, cranberry, cucumber, damson, date, dragonfruit (or pitaya), durian, elderberry, feijoa, fig, goji berry, gooseberry, grape, raisin, grapefruit, guava, honeyberry, huckleberry, jabuticaba, jackfruit, jambul, japanese plum, jostaberry, jujube, juniper berry, kiwano (horned melon), kiwifruit, kumquat, lemon, lime, loquat, longan, lychee, mango, mangosteen, marionberry, melon, cantaloupe, honeydew, watermelon, miracle fruit, mulberry, nectarine, nance, olive, orange, blood orange, clementine, mandarine, tangerine, papaya

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess/blind

COLOR- AND PIECE-BLIND CHESS

DR. TOM MURPHY VII PH.D.

1. Impressing humans

What better way for humans to impress each other with their brains, especially in movies, than to play chess—and to shout dramatically CHECKMATE! upon surprise-checkmating their opponent? Well, one way is to play chess while disadvantaged somehow, for example, by punching each other in the face repeatedly during the game to impair brain function (see Chess Boxing [8]). Another common distraction is to play a multitude of games against many opponents at the same time, in a so-called "simultaneous exhibition." The idea is that this is more challenging because of the need to maintain mental state for so many games at once, whereas your opponents only need to maintain state for one game. In truth, simultaneous exhibitions easily fall to a "man-in-the-middle attack." If the purported genius simply creates a perfect bipartite matching of the games played with the white pieces and the games played with black, he can mechanically forward moves between these pairs of boards. This requires only constant state (see next section) per pair of games, and guarantees an even score for the exhibition. So that's not very impressive.

Another disadvantage that humans sometimes use to impress each other is a blindfold (worn over the eyes). In this predicament they only hear the opponent announce moves and must imagine the position on the board in their mind's eye, both for the sake of remembering it and while exploring potential moves. Disadvantages can be combined, such as in the final scene of the 1988 documentary *Bloodsport* where Jean Claude van Damme is blinded by an illicit foreign substance during the final martial art battle.[1]

2. Impressing computers

In contrast, it is much more difficult to impress computers or impress people with computers. When it comes to computers playing chess, largely, the jig is up; it is now easy for chess programs, running on consumer hardware, to defeat the strongest human players. It is well known that striking a computer actually *fixes* it, so Chess Boxing becomes trivial. Blindfold chess is the natural interface for a chess computer; it is actually *much more difficult* to have the computer interpret the opponent's move by visually studying a physical board!

Playing multiple games simutaneously is an easy extension of playing a single game, although in principle the scale of such a thing could still be impressive. This is also impressive to other computers, who are largely concerned with filling up their memories with efficiently coded data. With a modern chess engine, it is easy to scale to an arbitrary number of games, since the exhibitor can make progress by observing one of the boards, computing a strong move, and playing it; this requires $O(1)$ space because all of the state is stored externally in the exhibition itself. However, we run the risk of losing the tournament (other players may be yet stronger computers). The man-in-the-middle attack remains an efficient way to minimize loss (ensuring an exactly even score). The simplest way to do this is to explicily generate a perfect bipartite matching over the n games G being played. This consists of $n/2$ pairs $\langle G_w, G_b \rangle$ (where we play as white against Bob and black against Walice, respectively). Since each game starts in the starting position, this is very easy; we can just assign the matches consecutively. Along with each pair we also record which of the following states we are in:

(1) We are waiting for a move from Walice (our white opponent)
(2) We have seen a move from Walice, which is ______.
(3) We are waiting for a move from Bob (our black opponent)
(4) We have seen a move from Bob, which is ____________.

If in State 1, we just watch G_b until Walice makes a move, then record it and proceed to State 2. We consume the move and move to State 3 by playing that move in G_w against Bob (where it must be our turn). We can immediately seek out that game or wait until we naturally come upon it. However, we should only approach G_w when the pair of games is in State 3, etc., otherwise we will not have a move to play.

There are $n/2$ pairs, with two bits for the state, no more[2] than $\log_2(64 \times 64 \times 4) = 14$ bits for each move (source square, destination square, and 2 bits to distinguish promotion to queen, rook, bishop, or knight). However, we also need to store the matching of G_w to G_b; this can be done with a pair of indices (or e.g. memory addresses) but unfortunately, this requires $\log_2(n)$ bits to represent. So overall this approach requires $O(n \log(n))$ space to play a simultaneous exhibition of n games.

It appears to be possible to reduce the space usage per game to a constant. In order to perform a man-in-the-middle attack, we need a perfect matching between the white games and black games. It is not essential that the matching be stable over time; for example if we are forwarding moves between Walice and Bob, and between Waluigi and Bario, and these games happen to transpose to the same position, then it works just fine to switch to forwarding between Walice and Bario; Waluigi and Bob. So, rather than store the matching explicitly, we can reconstruct it from the stored state at each step.

Let's think about the process of forwarding the moves from our white opponents to our black opponents; the reverse is of course symmetric. The first step will be to wait for the

Date: 1 April 2019.
Copyright © 2019 the Regents of the Wikiplia Foundation. Appears in The Journal Of LaTeX Class Files with the insufficient material of the Association for Computational Heresy; *IEEEEEE!* press, Verlag-Verlag volume no. 0x40-2A. 1 tempo.

[1] JCVD does not play chess on camera, but it is implied that he is also holding a simultaneous exhibition between rounds in a different room of the underground Hong Kong illegal karate complex.

[2] There are only 1792 pairs of squares between which pieces can ever move (Section 5.3.1), so $11 + 2$ bits suffices, with some added complexity.

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess/blind

white opponents to make their moves, and then copy these $n/2$ positions (not moves) into a vector in memory.

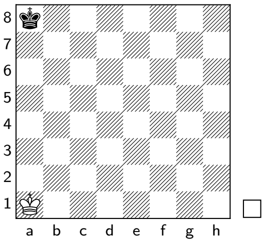
The black opponents are waiting for a move from us. Next, we'll copy their $n/2$ positions into memory, aligned with the $n/2$ positions already present. Let's say that the relation that defines a legal move in chess is

$$B \overset{m}{\to} B'$$

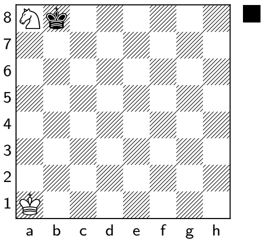
where B is the position before the move m, and B' is the resulting position. Our goal is to align the games such that B' (a position copied from our white opponent) is aligned with B (a position pending a move for our black opponent) in memory; this will represent the perfect matching. Computing m from B and B' when $B \overset{m}{\to} B'$ is easy (and unique), so this allows us to read off and play the move for each row.

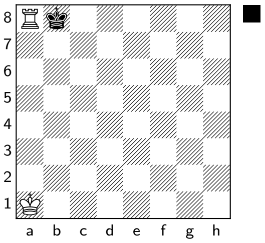
By invariant, it will be possible to produce such an alignment. For example, the first time we do this, each B will be the starting position, and B' will be a legal move made by white from the starting position. Any alignment will work. Let's say that just one of the white opponents played 1. d4, resulting in B_0; then one black opponent will make a legal response to this (say 1. ...Nf6, giving B_1). Then B_1 can be B' for the next round, which we can align with $B = B_0$, and so on.

The only tricky thing is figuring out which boards go with which. Although if $B \overset{m}{\to} B'$ it is easy to deduce m, it is not possible to compute B from B', or even from B' and m. This is because we may have both $B_a \overset{m_a}{\to} B'$ and $B_b \overset{m_b}{\to} B'$ with $B_a \neq B_b$. For example with B'

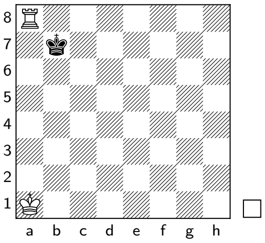


...we could have B_a and B_b be





Both of which can precede B' (the move is even the same: Kxa1). So it is not enough to greedily assign edges in our perfect match; if we choose the edge B_b to go with B', we might later find B'_2:



...and have no possible matching board, since it cannot legally follow B_a.

Fortunately, we know that there exists a matching (assuming all players are playing legal move e can tell if we found one (i.e., we didn't get stuck). So, one strategy that works is to choose randomly when there is some ambiguity, and start over from the beginning if we ever get stuck. In practice this will be pretty efficient, since convergent board states are unusual. We only need a single pseudorandom pool for the entire process, so it can be $O(n)$ bits; this seems easily enough to generate all possible permutations of $n/2$ items. Even 2^{2n} grows much faster than $n!$. If we don't like the random approach, I believe it is also possible to compute `next_permutation` in constant space; so we can just explicitly try all orderings for B' (this takes exponential time).

Once we have paired up each B and B', we simply compute the move (which we now know exists) and play it in that game. We then wait for the black opponents to play their moves, copy the resulting board states into our vector and repeat the above process (but flipping "black" and "white").

Although this is more involved than the previous approach, and may take exponential time, it allows us to play against n simultaneous opponents using $O(n)$ space!

2.1. **Board representations.** The actual space used per game is primarily the representation of a chess position, plus a few bits for bookkeeping. So, representing boards compactly gives us a way to increase the number of simultaneous games we can play for a given storage size.

Mainly, we need to store the pieces and their locations. There are a few other bits, like whose turn it is (1 bit), whether each player can still castle king- and queen-side (4 bits), and whether and where an en passant capture is possible (4 bits).[3]

With 64 squares, six pieces for each of the two sides, plus the empty square, a straightforward representation of the board uses $64 \times 4 = 256$ bits. The Thursd'z Institute considered more compact representations [2]; one nice choice works as follows:

Start with a single 64-bit mask which indicates, for each square on the board, whether it contains any piece. Note that there can only be up to 32 pieces on the board. To tell what these pieces are, we then follow with 32 four-bit records; these indicate the piece's color and type.[5] With the 9 bits of extra

[3] Technically, we need to store a lot of additional information with the board in order to completely implement the rules of chess.[1] The trickiest of these involve the rules for draw by repetition, which make reference to the history of the game (See Footnote 1 in *Survival in chessland* [6]) and seem to require storing all previous board states. Fortunately, if we are being this picky, then we also know that the length of a chess game is bounded by a constant: Rule 9.6.2 ends the game in a draw if both players have made 75 moves without a pawn move or capture,[4] so it suffices to store the 75×2 most recent (half-)moves. This sucks so most people don't do it (for example, FEN notation only gives the number of such moves, and so cannot implement the draw by repetition). On the other hand, if we insist, then this may give us a simpler route to a constant-space exhibition, since the $B \overset{m}{\to} B'$ relation is probably reversible with such information.

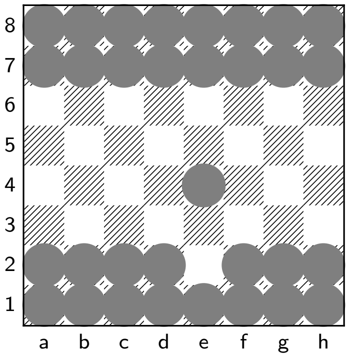
[4] These are two types of moves that make it impossible to formally repeat a position that preceded them. Castling also has this property, but doesn't count because it is a "secret move."

[5] Since only 32 of the 64 bits can be set, you could do slightly better by representing $\binom{64}{32}$ in ~ 61 bits. When fewer than 32 squares are occupied, we can use a record containing e.g. a third king (otherwise impossible) to indicate that we should ignore the remaining bits. However, this gets vastly more complicated for only 3 bits of savings.

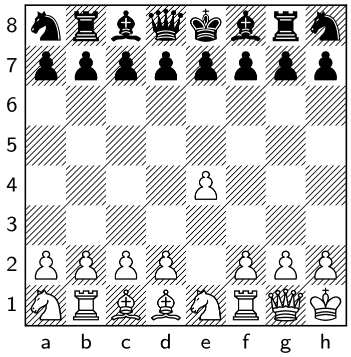
https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess/blind

state above, this totals $64 + 32 \times 4 + 9 = 201$ bits. There is some slack in the representation, because there are only 12 actual possibilities for a piece but we can indicate 16 with 4 bits. It is great to abuse this slack to save bits; for example, we can store a new type of rook that is still able to castle (it can only be in the corners and thus also indicates its own color), eliminating the 4 castling bits. We can similarly introduce an en-passantable pawn, saving the 4 bits of en passant state; this piece can only be in ranks 4 or 5, so it also indicates its color. We can also have a "king whose turn it is" for each side, saving the side-to-move bit. This totals a nice round $64+32\times 4 = 192$ bits.[6] This would allow approximately an 11 billion-game simultaneous exhibition in RAM on my desktop computer.

So now comes the main idea of the paper, which is also spoilered in the very clear paper title. What if we represented boards *only* as the 64-bit mask telling us what squares are occupied? The encoding is very lossy, of course, but it often contains enough information to deduce the state of the board. For example, can you tell what position this is?



Correct! It is



after 1. Nf3 Nf6 2. e4 Ng4 3. Bc4 Ne5 4. O-O Ng6 5. Kh1 Rg8 6. Nc3 Nh8 7. Qe2 Nc6 8. Rb1 Rb8 9. Nb5 Nb4 10. Rd1 Nd5 11. Qf1 Nb6 12. Qg1 Na8 13. Nbd4 Nb6 14. Rf1 Na8 15. Be2 Ng6 16. Bd1 Nh8 17. Nb3 Ng6 18. Ne1 Nh8 19. Na1

3. Color- and piece-blind chess

So this is kind of like blindfold chess, but for computers! Instead of being blind to the board, and only relying on our memory (trivial for computers), we'll only be able to see where the pieces are positioned on the board, but not what type or color they are. Of course, we also have to prohibit the computer from simply remembering the board, so the algorithm must be stateless. Specifically, we want a function

$$\texttt{makemove : uint64} \rightarrow \texttt{move list}$$

that makes a move from a single board position, represented just as 64 bits. This is a single move; the move list represents our preference for the move to make in descending order, and we commit to the first move that is actually legal. It does not work well to insist that the function return a single move, as it will often be the case that the board is misinterpreted and a move is in fact illegal; forcing forfeit[7] would mean that almost all games end in forfeit, which is boring. On the other hand, allowing the function to try again upon learning a move is illegal would allow it to interactively "interrogate" the board state somewhat.[8] This seems counter to the spirit of color- and piece-blind chess, so we instead require the function to rank all moves ahead of time.

4. Unblinding the board

I went about this by building a function that "unblinds" a board; it has type

$$\texttt{unblind : uint64} \rightarrow \texttt{position}$$

This function is natural for machine learning. It is easy to generate copious training data from actual games by simply blinding positions into their corresponding 64-bit numbers; I just randomly sampled 100 million positions from Feburary 2017 on lichess.org.

I repurposed the custom neural network code from my seminal paper *Red i removal with artificial retinal networks* [3] after discovering that artificial retinal networks are actually *isomorphic* to neural networks. The main advantage of this code is that it allows for sparse networks, but the real reason to use it is that I would rather spend dozens of hours debugging my own code, pay a larger electric bill, and get worse results in the end, than to spend a short while trying to get someone else's probably-very-good neural network training package to compile.

The structure of the network is as follows. The input layer is 64 nodes, one for each of the 64 bits, with each node set to either `1.0f` or `0.0f`. Three hidden layers of size 1024, 12288, and 567 do the neural magic. The output layer is 837 nodes; the bulk of which is a "one-hot" representation of the predictions for the 64 squares, each with 13 possible contents (black or white, six pieces, or empty). This is $64 \times 13 = 832$ nodes. Then four nodes to predict the four castling bits, and one to predict the current side to move. This model does not predict the possibility for *en passant* capture, nor move counters or position repetition. This will not be its main source of disadvantage!

I trained the network in two phases, first starting with a densely connected one (model size 160 MB), and then after I get fed up with how slow training was, a "vacuumed" version of the network where I removed edges with low absolute weights

[6] Since this is SIGBOVIK, I am freed from the burden of comparing related work. I did however read the rather bad Wikipedia article on the topic [7] which describes a Huffman-based encoding that uses a "maximum of 204 bits, and often much less." This also includes a 7-bit 50-move counter (but you really need to implement a 75-move counter; 50 moves only allow a player to *claim* a draw) so should be compared as 197 bits. But the article also contains many bugs, like the misconception that there can only be four total rooks (pawns are allowed to promote to rook). So the approach described here is both *more efficient* and *more correct*.

[7] FIDE rules state that the *second* attempt at an illegal move results in forfeit (7.5.5).

[8] Similar to Kriegspiel [9], although in that game at least one's own pieces are known!

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess/blind

(model size 5 MB) to continue training. Removing edges based on an absolute weight threshold is very unprincipled (since a downstream edge can magnify a contribution arbitrarily) but I did it anyway.

Everything was trained on a single GPU, though a fairly decent one in 2018, the EVGA GeForce GTX 1080 “FTW” (really), using OpenCL. Biases were initialized to 0 and weights with a Gaussian of mean 0 and standard deviation 0.025. In the first phase, there were 64 examples per round, and after vacuuming, 2048. The round learning rate α_r started at 0.1 and descended linearly to 0.002 over 500,000 rounds; and the learning rate when updating weights (for each example) α is α_r/`examples_per_round`. In no way am I recommending these parameters. Fiddling with the parameters to make it do its black magic or alternately carom off to a sea of NaNs or zeroes is for sure the worst part of neural networks. Indeed, I initially started with the classical sigmoid transfer function, but “upgraded” to the “leaky rectified linear” function

$$(p < 0) \mathbin{?} p \times 0.01 : p$$

after getting fed up with sigmoid weights caroming off (see “vanishing gradient problem” and/or “exploding gradient problem”). The final model was trained over 339,885 rounds on 223 million examples. It did not appear to show signs of improving for several days before I terminated it.

4.1. **Statistical evaluation.** The unblinding component can be evaluated on its own, by running it on examples sampled independently of the training set. The model outputs a score for each possible contents of each square; we simply discretize to the highest-scoring one (same too for the predicted castling state and turn). Over 50,000 examples, these were the results:

9,584 predicted boards were exactly correct (19.17%). There were a total of 161,166 piece mistakes, which is an average of 3.22 per board. This is wherever we predict a square’s contents incorrectly. There were only 1630 castling status mistakes, an average of 0.03 per board (there can be up to four mistakes per board). This is probably because when the king and rook are in their starting squares, castling is almost always still allowed. In 19,014 boards, we mispredicted whose move it is (38%). This appears to be the most difficult thing to predict, which is not surprising.[9]

4.2. **Subjective evaluation.** The unblinder *must* make mistakes since the 64-bit representation is ambiguous. Subjectively, the unblinder makes reasonable mistakes. It is excellent at common openings, usually getting these exactly correct. On the other hand, it is fairly bad at sparse endgames, where it is difficult to tell a pawn from a rook from a king. It is terrible at unlikely positions that can be confused for likely ones. If you are playing against it and know how it works, it is easy to trick it by doing something like capturing one of its starting-position pawns with your queen; nobody does this in real games (because the queen can be immediately recaptured), so the square is predicted as a pawn and the queen “disap[illegible]o the unblinder (Figure 1). Having an invisible queen [illegible]camp is of course very dangerous. Resolving ambiguities in favor of more likely positions is the right thing for the model to do, so this is just an inherent flaw with the decomposition of the problem. There are some ways we can account for this (Section 5.2).

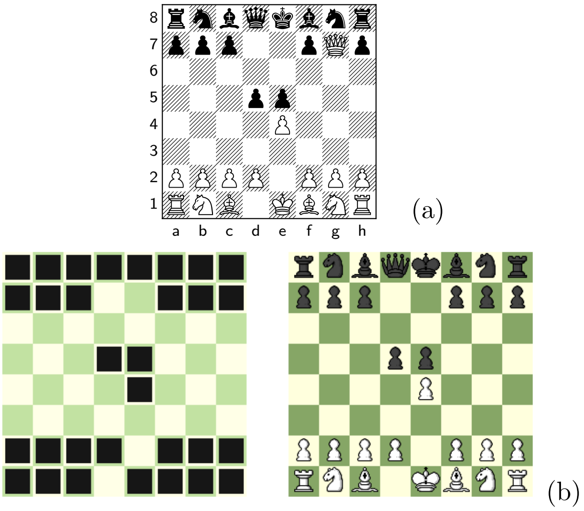


Figure 1. (a) Position after 1. e4 e5 2. Qg4 d5 3. Qxg7; note the white queen strangely on g7. (b) The bitmask for this position and the unblinder’s prediction. The queen “disappears” after Qxg7, because unblinding predicts it to be one of black’s own pawns—far more likely in that square.

A few things are distinctly disappointing about its performance. Even outside of “likely” positions, it usually predicts that pieces on black’s side of the board are black, and vice versa (Figure 2). This makes sense, but suggests serious limitation on using the prediction to play chess. Less forgivably, it sometimes predicts more than one king per side (or zero), which is always wrong. Actually, an early version had this problem in spades, frequently predicting two or three kings. Upon debugging, I had simply made the noob mistake of printing both King and Knight with the letter “K.” Ha! It often predicts the “wrong” number of bishops (etc.), or places them on the same color. This is technically possible through promotion, but usually a bad guess, since promotion is relatively rare, and moreso promotion to a piece other than a queen. An approach that might handle this better (but may have different downsides) would be instead to predict the “fates” of the 32 initial pieces [6]. The fate of a pawn includes what if any piece it has promoted to, but this is not necessary for the other pieces. This would *require* that the model only predict a single location for each king, among other things. However, this would require a much larger output layer (32 pieces can move to 64 squares, plus promotion) and it is not always clear how to interpret its value into a position (for example, if two pieces are predicted strongly to be on the same square).

5. Playing blind

Once we have predicted a board state, we can play with it. The simplest way to do this is to use a strong chess engine to

[9] Prior to “vacuuming”, the 160 MB network actually performed slightly better than the final 5 MB network, with 21.20% of boards exactly correct, and an average of 3.12 mistakes per board. This suggests that the model may only be doing a limited amount of generalization, instead mostly memorizing board positions. Representing the 223 million examples seen exactly (using our best board representation described in Section 2.1) would take 42.8 GB, so at 5 MB at least the data is represented compactly, if also rather lossily.

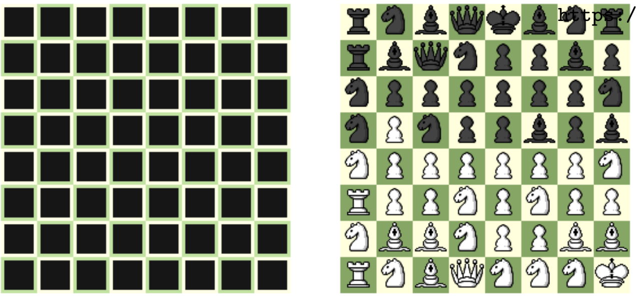


FIGURE 2. What the model predicts (in single-king mode; Section 5.1) for 0xFFFFFFFFFFFFFFFF, the board with all bits set. This is an impossible position, but it gives some idea of the model's biases for each square. Notably, most of the pieces on each half of the board have a single color. This makes sense, but also suggests substantial limitation. When single-king mode is off, the bottom right king is predicted as a white rook.

pick a move for the position. Here, I use Stockfish with a node budget of 1 million nodes, which takes about 1 second of CPU time per move on my computer. There are some complications:

- Frequently, the unblinded board will not match the true position, and Stockfish will choose a move that is illegal. So, as discussed before, we actually return a prioritized list of moves. For this first experiment, we just return the move Stockfish recommends, followed by all other moves ordered randomly.
- Stockfish is a very strong engine, and in my opinion the code is generally good, but it is very sensitive to bad FEN (the notation used to give it a position to evaluate) strings. Given a bad string, like one that says castling is possible when the king isn't in its home square, often crashes the engine. So we need to make sure to actually pass valid positions. I accomplish this by making the following modifications to the predicted board:
  - If a castling bit is set, but castling is not possible due to the position of the king or rook, clear the bit.
  - Set the side-to-move to be the correct actual value. This uses the unblinded state, so is superficially cheating. But note that if we get the side wrong, then Stockfish's move will always be illegal: Moves are specified as a source and destination square,[10] and so the source square of Stockfish's move would always be a piece of the wrong piece's color. So this is equivalent to (but twice as efficient as) running stockfish twice, one for each side, and prioritizing the predicted side's move first.

  This won't fix all positions, for example, if the white and black king are adjacent to one another in mutual check. If an irreparable problem is detected, then I just return a uniformly random move list.

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess/blind

It is easy to beat this chess engine, by ; it as in Figure 1, although this involves unnatural so it may only apply if you know how it works. Meas ow well it plays in an absolute sense is a subject of some interest, so I wrote a separate paper about that [5]. This algorithm, called `blind_yolo`, had an Elo World score of 489 ± 2. It beats a purely random player with a score of 101 wins, 27 losses, and 389 draws. Making moves purely at random is one of the few fair comparisons, since the random strategy also works with color- and piece-blind chess.

5.1. **We three kings.** When evaluating the first version I found that it was predicting a disappointingly high number of illegal positions in practice, which was causing us to fall back on making random moves, which is mostly boring. The second version reduces the rate of illegal positions due to too many or too few kings [4].

The model predicts a score for each type of piece in each square, and we do not have to necessarily interpret it by always taking the highest-scoring piece. This version first finds the two squares with the highest scores for the white king, and same for the black king. We take two in case the same square is predicted for both. Then this square gets one of the kings (whichever has higher score) and the other king goes in the highest-scoring unclaimed square. The rest of the squares get the highest-scoring prediction as before, but we never predict kings for them.

This change just affects the unblinding process, so we can directly evaluate its accuracy. It gets 19.28% of positions exactly correct (slightly better), with an average of 3.26 piece mistakes per position (slightly worse). This is expected; we exchange local mistakes (each was trained independently to minimize its local error) for global correctness (which is not taken into account at all during training).

This version, called `blind_kings`, performs a small amount better than `blind_yolo` (63 wins, 45 losses, 412 draws). It had an Elo World score of 502 ± 3.

5.2. **Spy check.** Say `blind_kings` is playing as black; it remains easy to fool it by moving black pieces into white's camp, since they are usually then predicted to be white pieces. We can defend against this somewhat. Since it is illegal to capture one's own piece, there is little risk in trying; if it is indeed our own piece then the move will be rejected, and if it is not our piece, then capturing is good for two reasons: One, we capture a piece, and two, we avoid having Stockfish make a move in this incorrectly predicted board. (Of course there are many reasons why eagerly capturing a piece can be a bad idea, but at this level of play, an edge in material is likely worth it.)

There is one subtlety here. Above we argued that it was safe to use the actual side-to-move instead of the predicted one; but here it would not be equivalent to do so. Instead, we first prioritize all apparent spy-check moves where the predicted source piece matches the predicted side-to-move, then we try the opposite. (Ties are broken by preferring to capture with a lower-value predicted piece, and then randomly.) Due to this, there is some additional chance that we end up making an especially dumb move because we both mispredicted the side-to-move and the identity of some pieces.

This version, `blind_spycheck`, works significantly better than `blind_kings`. It has an Elo World score of 547 ± 1,

[10] Plus promotion piece. Castling is represented as a two-square move of the king.

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/chess/blind

somewhere between a 93.75–96.875% dilution of stockfish1m (the third best non-engine player).

5.3. **Future work.** The predicted board often expresses uncertainty about some squares, which could be thought of as probabilities. A principled improvement would be to try to find moves that are good in expectation, that is, integrated over all possible boards in proportion to their predicted probability. A good approximation might be had by sampling a bunch of boards according to the predicted distribution, and then using Stockfish to score the top k moves for each; we can then order moves by their expected score. Unfortunately, it is not easy to efficiently get Stockfish to generate scored moves for $k \neq 1$. Even with $k = 1$, this approach would be slow, taking about a second for each (distinct) sampled board. So I did not try it, at least not before submitting this paper.

5.3.1. *No,* u r *a lnetwork.* I initially considered trying to solve this whole problem with neural networks. The current best known engine in the world (AlphaZero) at least *uses* a neural network. The biggest advantage would be that it would naturally be able to consider multiple moves under uncertainty about the board state, as just discussed, without any particular extra logic. My plan was to make multiple different components that could be evaluated separately, starting with the unblinder described, followed by a unit that predicts legal moves, and then a unit that takes these two (and also the 64-bit blinded mask if it likes) and scores each move. Predicting a legal move is also a natural function for machine learning; a move can be given just as a source and destination square.[11] Many pairs of squares are always impossible (e.g. no piece can ever move from A1 to B8); so there are only 1792 potential moves to predict. However, training a reasonable unblinder took longer than I expected, and the legal move predictor never really worked that well (it has a harder job), so I just settled for basing it off the single unblinder unit. Can you do better?

6. Conclusion

I would like to thank the little people (pawns) and the the author- and content-blind anonymous referrees.

References

[1] FIDE handbook – E.I.01A. Laws of chess, 2017. www.fide.com/component/handbook.

[2] Jim McCann, David Renshaw, Ben Blum, William Lovas, and Tom Murphy, VII. Chessboard representations, December 2018.

[3] Tom Murphy, VII. Red i removal with artificial retina networks. In *A record of the proceedings of SIGBOVIK 2015*, pages 27–32. ACH, April 2015. sigbovik.org/2015.

[4] Tom Murphy, VII. CVE-2018-90017117. In *A Record of the Proceedings of SIGBOVIK 2019*. ACH, April 2019.

[5] Tom Murphy, VII. Elo World: A framework for benchmarking weak chess algorithms. In *A Record of the Proceedings of SIGBOVIK 2019*. ACH, April 2019.

[6] Tom Murphy, VII. Survival in chesslan *Record of the Proceedings of SIGBOVIK 2019*. AC 2019.

[7] Wikipedia. Board representation (chess). https://en.wikipedia.org/wiki/Board_representation_(chess).

[8] Wikipedia. Chess boxing. http://en.wikipedia.org/wiki/Chess_boxing.

[9] Wikipedia. Kriegspiel (chess). https://en.wikipedia.org/wiki/Kriegspiel_(chess).

[11] There are also four choices for promotion when moving a pawn into the last rank. It is always the case that if any promotion is legal, all choices are legal, so this does not need to be encoded in this phase. Also, at this level of play, always promoting to queen is a very safe simplification.

Dimensionality-Reducing Encoding for Classification of Pythagorean Engendered Numbers

Dead Duck or Phoenix?

π, 2019

Rany Tith
Spark Tank
Riverside, CA, 92507
rany.tith@protonmail.com

Oscar I. Hernandez
ARKS
Riverside, CA, 92507
ohernandez13@simons-rock.edu

Abstract

We apply modern well-known machine learning classifiers to the problem of determining whether a given positive integer is even or odd, a problem dating as far back as 6th century Classical Greece before common era and even further in Ancient Egypt. We prove that the classification of engendered numbers is possible. This is done by utilizing a unique dimensionality-reducing encoding that was implemented before the machine learning models were trained. Overall, our models' results proved to be successful in classification as indicated through AUC and ROC analysis.

1. Introduction

In this article, we'll discuss a partition of numbers discovered by the Ionian Greek philosopher Pythagoras (c.570–c.495 BC), which he discovered during his tenure in Egypt before founding a school of math in Greece. "To him, the odd numbers were male, and the evens were female"[PYT].

Following Pythagoras, we will restrict our attention from the usual integers, $\mathbb{Z} = \{..., -1, 0, 1, ...\}$, to only the positive integers, $\mathbb{Z}^+ = \{1, 2, 3, ...\} \subset \mathbb{Z}$. Although one can evaluate whether a small number is even or odd, the general problem remains open.[1] In the following sections, we will formally define even and odd numbers, build a classifier, evaluate its results in a case study, discuss potential improvements, and conclude with relevant open problems.

2. Preliminaries

2.1 Even and Odd

Definion 1. *[MNT] Let $n \in \mathbb{Z}^+$ be a positive integer. We say that n is even* iff[2] *$\exists k \in \mathbb{Z}$ such that $2 \cdot k = n$ Similarly, n is odd* iff *$\exists k \in \mathbb{Z}$ such that $(2 \cdot k) + 1 = n$.*

For example, $1 = 2 \cdot 0 + 1, 3 = 2 \cdot 1 + 1, 5 = 2 \cdot 2 + 1, 9 = 2 \cdot 4 + 1, 7 = 2 \cdot 3 + 1$ are odd and $2 = 2 \cdot 1, 4 = 2 \cdot, 6 = 2 \cdot 3, 8 = 2 \cdot 4, 10 = 2 \cdot 5$ are even. We have manually classified the next 60 numbers. In the next section, we will use that labeled data to train a classifier to evaluate the even-ness/odd-ness of a given number.

2.2 Receiver Operating Characteristics [ROC]

This section is pulled directly from the source cited. A receiver operating characteristics (ROC) graph is a technique for visualizing, organizing, and selecting classifiers based on their performance.

Definion 2. *ROC graphs are two-dimensional graphs in which tp (true positive) rate is plotted on the Y axis and fp (false positive) is plotted on the X axis.*

They depict relative tradeoffs between benefits (tp) and costs (fp). We'll see the tradeoffs in Section 4.

3. Solution

We propose to use real data such as seen in Figure 5 to train machine learning classifies using the Python library `scikit-learn`. In particular, we will implement support vector machines (SVM), multi-layer perceptrons (MLP), decision tree classifiers (DTC), as shown in Figure 1.

The code is freely available at the first author's GitHub repository. [DRECPEN]. All the results achieved in section 4 were performed on a x86-64 Arch Linux Thinkpad running Python 3.7.2.

3.1 Encoding

In particular, we pre-process the information by encoding the positive integer in binary [AB]. We optimized this encoding and reduced its size to a single bit by restricting our data to the rightmost bit and disregarding the rest of the binary string, since memory becomes a concern when dealing with large numbers. The authors are not in agreement as to why this yields such accurate results, but they are proud of its size and speed.

[1] The authors have attempted to communicate with him, but have been informed that he is not an advocate or user of computers or electronic mail.

[2] if and only if

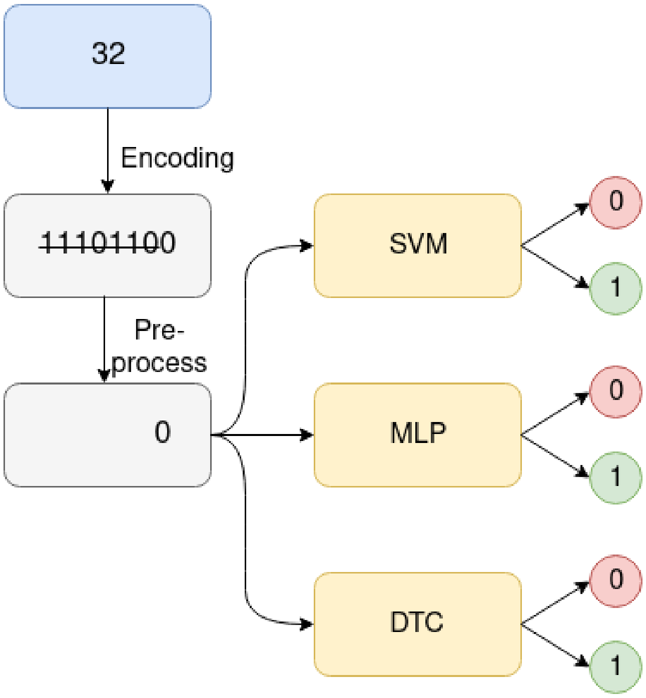


Figure 1. Flow diagram of classification process

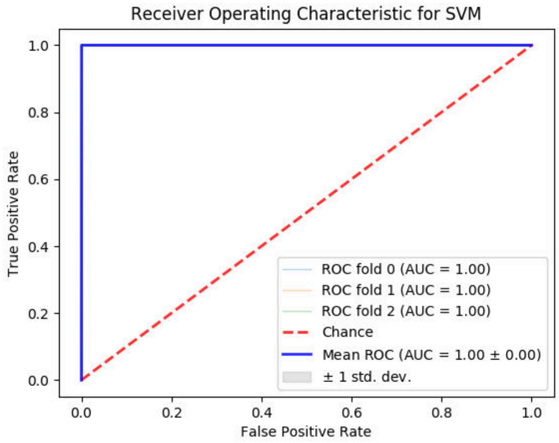


Figure 2. SVM ROC analysis

4. Case Study: Classification Models

4.1 SVM

4.1.1 Discussion

SVM is a popular machine learning model created by Vapnik and Chervonenkis in 1963 which has been implemented successfully in areas such as image classification and hand written character recognizition. The SVM method employs to minimize the equation[SVM]:

$$[1/n\sum_{i=1}^{n} max(0,1-y_i(w\cdot x_i-b))]+\lambda|w|^2$$

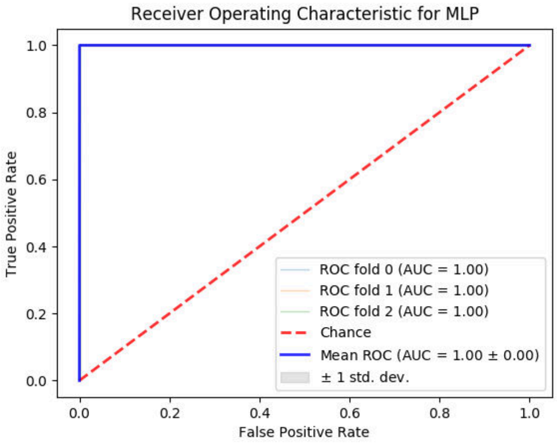


Figure 3. MLP ROC analysis

where w denotes the weight, x is the input variable, y is the dependent variable and λ indicates the margin strength for classification.

4.1.2 Evaluation

In our evaluation as seen in Figure 2 ROC analysis indicates a strong informative model. This is shown as the mean ROC is above the chance line with a low standard deviation. Furthermore, the AUC indicates 1.00 leaving us to conclude that it does better than random chance.

4.2 MLP

4.2.1 Discussion

MLP models are considered to be a type of deep learning that has found to be useful in fields such as speech recognition and image recognition. The model employs layers of neuron that contains the following ReLU activation function:

$$f(x)=max(0,x)$$

Each layer in the neural network contains a number of nodes where a weight w_{ij} was applied at each level to each node.

Learning was then done using back propagation such that:

$$e_j(n)=d_j(n)-y_j(n)\epsilon(n)=1/2\sum_j e_j^2(n)$$

d is the target value, y is the output of the perceptron.

And gradient descent was then used to optimize the weights:

$$\Delta w_{ji}(n)=-\nu\frac{\delta\epsilon(n)}{\delta v_j(n)}y_i(n)$$

Where ν is the learning rate, v_j is the sum of all the nodes input, y_i is the output of the previous neuron[MLP], n is the data point, j is the position of the outpude node, and e is the error.

4.2.2 Evaluation

In our evaluation as seen in Figure 3 ROC analysis indicates a strong informative model. This is shown as the mean ROC is above the chance line with a low standard deviation. Furthermore, the AUC indicates 1.00 leaving us to conclude that it does better than random chance.

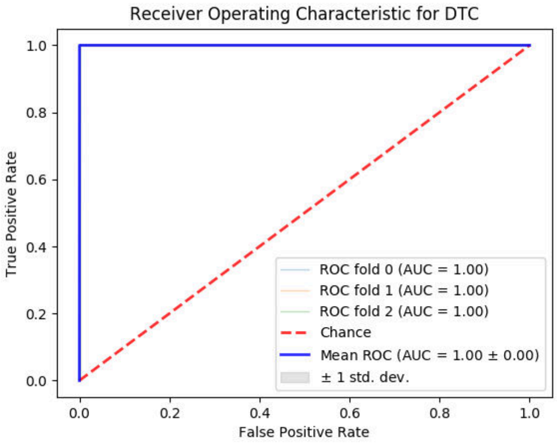


Figure 4. DTC ROC analysis

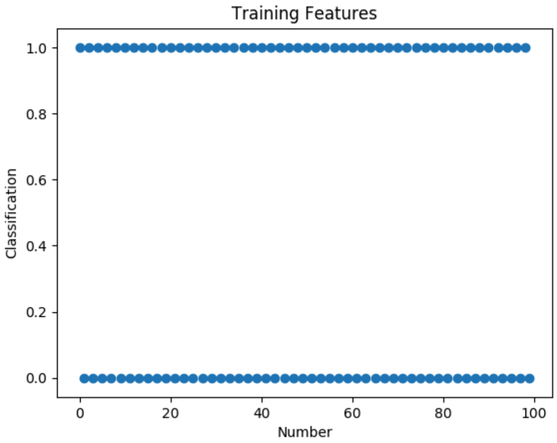


Figure 5. Example training data

4.3 DTC

4.3.1 Discussion

DTC is a popular white model method in use cases such as fraud detection, direct marketing, and economics. More generally we used a decision tree learning method with the information gain metric of:

$$H(T) = I_E(p_1, p_2, ..., p_j) = -\sum_{i=1}^{J} p_i log_2 p_i$$

where p_i are the fractionals that add up to the class that is present in each node that happens at each split in the tree[DTC].

4.3.2 Evaluation

In our evaluation as seen in Figure 4 ROC analysis indicates a strong informative model. This is shown as the mean ROC is above the chance line with a low standard deviation. Furthermore, the AUC indicates 1.00 leaving us to conclude that it does better than random chance.

5. Discussion

The results are truly surprising. They prove that it is possible to classify numbers as being even or odd, a skill that even intelligent machines such as humans do not acquire naturally nor easily. Perhaps this issue is deeply tied to the fact that numbers, as Pythagoreas believed, were engendered, and humans often struggle with evaluating gender while neural networks (e.g. convolutional) find great success.

6. Conclusion

Although this problem is difficult (perhaps intractable), similar problems may be solvable. It may be worthwhile to classify prime numbers, Mersenne primes, perfect numbers, positive numbers, negative numbers, zero, and powers of two.

6.1 Acknowledgements

We would like to thank our colleague, a student at the Washington Elementary School for classifying the first 70 numbers as even or odd. This work was partially supported by Spark Tank, who provided us with coworking space and toiletries. Most importantly, we would like to thank our families, without which we could have completed this work 2 years sooner.

References

[ROC] T. Fawcett An Introduction to ROC Analysis *Elsevier Pattern Recognition Letters*. 2006. https://people.inf.elte.hu/kiss/11dwhdm/roc.pdf

[MNT] K. Ireland & M. Rosen A Classical Introduction to Modern Number Theory *Springer Graduate Texts in Mathematics*. 1990. https://www.springer.com/us/book/9780387973296

[DRECPEN] R. Tith & O. Hernandez Accompanying Source Code *GitHub*. 2019. github.com

[AB] G. Leibniz Explication de l'Arithmtique Binaire *Die Mathematische Schriften*. 1703. http://www.leibniz-translations.com/binary.htm

[SVM] C. Cortes & V. Vapnik Support-vector networks *Springer Machine Learning*. 1995. https://link.springer.com/article/10.1007%2FBF00994018

[DTC] L. Breiman et al. Classification and regression trees *Wadsworth & BrookeCole Advanced Books & Software*. 1984. https://www.crcpress.com/Classification-and-Regression-Trees/Breiman-Friedman-Stone-Olshen/p/book/9780412048418

[MLP] F. Rosenblatt Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms *Springer Brain Theory* 1961. https://link.springer.com/chapter/10.1007/978-3-642-70911-1_20

[PYT] W. Burkert Lore and Science in Ancient Pythagoreanism *Harvard University Press*. 1972. http://www.hup.harvard.edu/catalog.php?isbn=978067453918

The authors assure us the materials will be available “if they get around to it”: `http://zifyoip.com/emojizip`

`emojizip`: A text compression system based on pictogram–kiloword equivalence

William Gunther
Google
wgunther@google.com

Brian Kell
Google
bkell@google.com

SIGBOVIK ’19
Carnegie Mellon University
April 1, 2019

Abstract

🚶🚶👷‍♀️

1 Introduction

Data compression is a well studied topic with many applications. However, existing methods suffer from several limitations.

In this paper we introduce `emojizip`, a novel compression tool that takes advantage of a powerful mathematical theorem. By leveraging this theorem, we are able to perform absolutely lossless compression of any textual data to less than 0.1% of its original size. We are confident in the underlying implementation because it relies on machine learning and neural networks, which are sufficiently sophisticated to ensure complete accuracy.

2 Background

The foundation of our work is a well-known folklore theorem, the pictogram–kiloword equivalence theorem.

Theorem 1 (Pictogram–kiloword equivalence theorem)**.** *A picture is worth a thousand words.*

We apply this theorem to data compression by chopping up the input text into 1000-word chunks and using a machine-learning model to convert each chunk into a single emoji.

2.1 Previous work

Early work in the field established that a picture is worth a word [1].

Previous papers in this prestigious conference series have established that a word is worth arbitrarily many words [2] (extending earlier work [3]), a word is worth 108,709 words [4, 5, 6], and 79 words are worth 17 words [7].

Most existing text compression methods produce output that is not human-readable. Recent work has addressed a similar problem for compiled C code [8]. Our work does the same for compressed text.

3 Implementation

Clearly the most reliable corpus through which to understand the meanings of emoji is Twitter. Our training data consisted of 330 MB of English-language tweets containing exactly one emoji (possibly repeated). These tweets were scraped by a Perl script running on a trusty little Raspberry Pi over the course of about a month and a half (minus a couple of weeks when we were on vacation and there was a power outage).

3.1 Compression

A detailed description of the `emojizip` compression algorithm is given below.

Algorithm 1 Detailed compression algorithm.

```
procedure EMOJIZIP COMPRESSION
    TensorFlow model ← tweet data
    text ← normalized text
    for all 1000-word chunks ∈ text do
        translation ← translation, translated chunk
    end for return translation
end procedure
```

As it turns out, with TensorFlow it is surprisingly easy to get a Raspberry Pi to run out of memory and freeze. Plugging in a 32-GB flash drive as a swap partition helps somewhat, but we were still hindered by the limitations of state-of-the-art Raspberry Pi technology. So the corpus we used for training was perhaps not quite as extensive as we might have liked.

The first trial run of the compressor converted "seeing you makes me sad" to 🇵🇼, the flag of Palau. Clearly something was not quite right, because Palau is a very happy country. After a bit of debugging, the second trial run converted "Trump" to 🇷🇺, the flag of Russia, which means everything was working correctly.

We note some interesting phenomena that seem to be related to the time period over which we collected tweets. For example, the United States Declaration of Independence [9] compresses to 🇱🇹✝️. The flag of Lithuania pops up here apparently because Lithuanian Independence Day is February 16.

As an example to demonstrate the power of our approach, Figure 1 shows the entire text of the King James Version of the Bible [10] compressed into just 720 emoji.

We recommend the file extension .🤐 for compressed emojizip output.

3.2 Decompression

Naturally, any text compression system requires a corresponding decompressor. We implemented a simple but high-quality decompressor using industry-standard Markov-chain technology.

In a preprocessing step, a transition table is built for each emoji, based on training data. Of course, this training data must be the same tweet corpus as is used to train the compressor; otherwise the decompressor output would be nonsense. The transition table for a given emoji gives, for each pair (w, w') of words that appear in some tweet with that emoji, the probability $\Pr(w' \mid w)$, i.e., the probability that w will be followed by w'. Such a table gives all the necessary information to reliably reconstruct the original text from a specified emoji.

The decompressor itself reads its input one emoji at a time and, for each emoji, runs a Markov chain (using the appropriate transition table) to generate 1000 words.

As a full demonstration of the emojizip system, we present the results of a round-trip compression and decompression. When the script of Abbott and Costello’s famous “Who’s on First?” comedy routine is given to the compressor, the output is 🚶🤷‍♀️. Naturally. By decompressing these emoji, we can recover the original script; see Figure 2. Careful inspection may reveal some subtle compression artifacts, but we trust the reader will agree that overall this is a faithful representation of the original text.

4 Conclusions and future work

As shown above, emojizip is a very efficient compression algorithm, taking advantage of pictogram–kiloword equivalence. It naturally invites a few areas for future work and improvements.

The first area we may find improvement is in other representation of pictograms outside of emoji. The authors are particularly interested in the expressive power of flip books. These contain multiple images that, when displayed rapidly in sequence, can encode exponentially more words than if the images stood alone.

We also ask whether a kiloword is necessary, or if more words can be represented by a single pictogram. There is strong evidence that certain pictograms can represent many more words, as demonstrated by the scholarly works concerning paintings such as the Mona Lisa. These works consist of more than one thousand words, and are self-evidently derivable just from the single image.

The authors assure us the materials will be available “if they get around to it”: `http://zifyoip.com/emojizip`

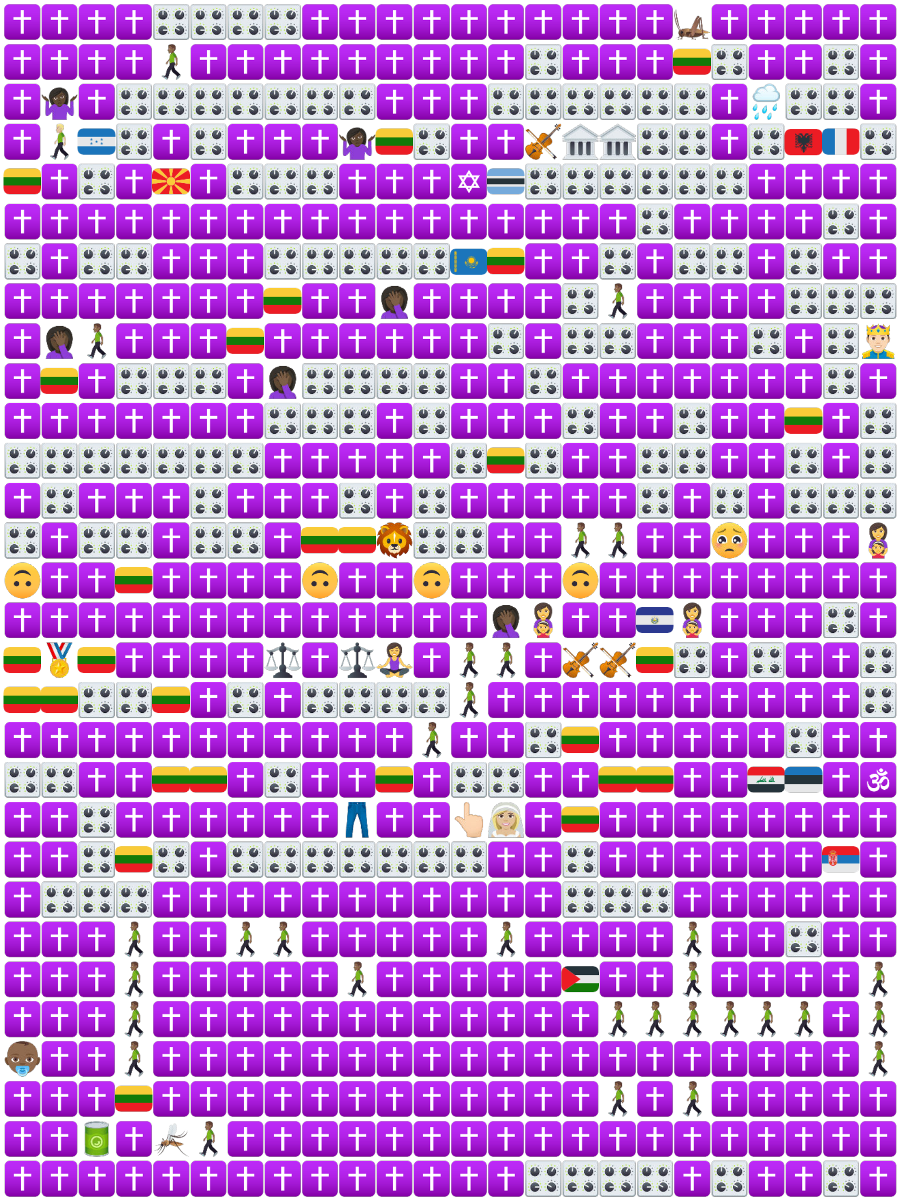


Figure 1: The Bible.

The authors assure us the materials will be available “if they get around to it”: http://zifyoip.com/emojizip

while y’all here are some things I go to hell” I go to you I’m extra single. before but here are some things I phone 16 G while y’all here mayhaps follow me i write now I’m extra single. This Emry is just an Arsene Wenger with black hair. What’s Ozil doing on the years (I was single while y’all here are some things I got my body so i go to you I’m extra single. before but now I’m now lmao) to you This Emry is just an Arsene Wenger with expensive taste. This Emry is just an Arsene Wenger with expensive taste. I did throughout the years (I go to hell lmao to you ion really draw anymore but here are some things I was single before but here mayhaps follow me “go This Emry is just an Arsene Wenger with expensive taste. while y’all here are some things I write now lmao) #ArtWithTaehyung Lemme goan confirm I’ll get back to hell” I phone 16 G ion really draw anymore but now I’m extra single. Stay away from poor girls with black hair. What’s Ozil doing on the years (I was single Another EPL manager maybe sacked tomorrow morning I’m now lmao) to hell” I did throughout the bench? Rubbish. I’m extra single. i don’t need nobody Hmm... Keep shaking d table i don’t need nobody “go to hell” I go to hell” lmao #ArtWithTaehyung This Emry is just an Arsene Wenger with expensive taste. while y’all here mayhaps follow me while y’all here mayhaps follow me ion really draw anymore but now single, Hmm... Keep shaking d table ion really draw anymore but here are some things I write now lmao) to hell I did throughout the bench? Rubbish. Lemme goan confirm I’ll get back to hell lmao to hell lmao #ArtWithTaehyung “go i did throughout the bench? Rubbish. I’m now I’m extra single. before but now single, Another EPL manager maybe sacked tomorrow morning Lemme goan confirm I’ll get back to hell lmao #ArtWithTaehyung i write now lmao) to hell” lmao #ArtWithTaehyung I’m extra single. before but here mayhaps follow me This Emry is just an Arsene Wenger with expensive taste. Stay away from poor girls with black hair. What’s Ozil doing on the bench? Rubbish. Stay away from poor girls with expensive taste. Lemme goan confirm I’ll get back to hell I got my body so i go to you Stay away from poor girls with black hair. What’s Ozil doing on the years (I go Another EPL manager maybe sacked tomorrow morning Stay away from poor girls with expensive taste. i did throughout the years (I did throughout the bench? Rubbish. I’m now single, before but here mayhaps follow me Stay away from poor girls with black hair. What’s Ozil doing on the years (I was single i write now I’m extra single. i go Hmm... Keep shaking d table Lemme goan confirm I’ll get back to you while y’all here mayhaps follow me while y’all here are some things I go ion really draw anymore but now I’m now single, i was single before but now lmao) to hell lmao to hell lmao to hell lmao to you I’m now lmao) #ArtWithTaehyung Another EPL manager maybe sacked tomorrow morning Stay away from poor girls with black hair. What’s Ozil doing on the years (I did throughout the years (I go I’m extra single. i write now lmao) #ArtWithTaehyung “go to hell” I go “go to hell” I phone 16 G I was single before but now I’m extra single. i go to you This Emry is just an Arsene Wenger with black hair. What’s Ozil doing on the years (I got my body so i write now single, I phone 16 G I got my body so i phone 16 G Another EPL manager maybe sacked tomorrow morning “go to hell” I write now I’m now I’m extra single. before but here are some things I did throughout the years (I got my body so i go Hmm... Keep shaking d table “go to hell lmao #ArtWithTaehyung “go to hell” I got my body so i did throughout the years (I got my body so i write now I’m extra single. i write now single, i don’t need nobody im jos gonna. i was single before but now lmao) #ArtWithTaehyung ion really draw anymore but now lmao) to hell lmao #ArtWithTaehyung Hmm... Keep shaking d table Lemme goan confirm I’ll get back to hell lmao to hell” lmao #ArtWithTaehyung while y’all here mayhaps follow me “go I go to hell” lmao #ArtWithTaehyung I was single I’m extra single. i go I’m extra single. i go im jos gonna. I’m now I’m now single, i did throughout the bench? Rubbish. I’m extra single. Stay away from poor girls with expensive taste. ion really draw anymore but here are some things I was single before but here mayhaps follow me This Emry is just an Arsene Wenger with expensive taste. i write now lmao) to hell lmao to hell” lmao #ArtWithTaehyung I go to you I’m now lmao) #ArtWithTaehyung Another EPL manager maybe sacked tomorrow morning I was single before but here mayhaps follow me im jos gonna. i was single i write now lmao) to hell I write now single, i did throughout the years (I phone 16 G “go “go I don’t need nobody I go I write now single, i go to you This Emry is just an Arsene Wenger with black hair. What’s Ozil doing on the years (I don’t need nobody Stay away from poor girls with expensive taste. This Emry is just an Arsene Wenger with expensive taste. Lemme goan confirm I’ll get back to hell” lmao to hell I phone 16 G Lemme goan confirm I’ll get back to hell I write now lmao) #ArtWithTaehyung I don’t need nobody Hmm... Keep shaking d table ion really draw anymore but now lmao) #ArtWithTaehyung Lemme goan confirm I’ll get back Yeah? I said!!! “ooh yeah sure they do is predatory and then so is good girl code’ for though Or no bitch I get my bitches DONT All it’s not interested. anymore in a farm. season Not everything with no, bitch I receive here? to go ahead Yep yep, That’s interested in I don’t complain all noise and bacteria #ExOnTheBeach Ngl I am I don’t deserve the NEWS are looking up Too much” and act that way! Blame you, got a president That’s just disappear every artist? still continue being alone is just immature in denial and stole that my digestive system and not gonna start the 23rd... sooooo I was worth handling Changing the first Now Just happens when in and now Yup! some people that Apparently I’m the U.S. trends related stuff First u right to be alone is what Maybe tomorrow. if he said “come to, be helped too like I’m paranoid or specific person not make everyone grew up to get past Heteronormativity isn’t any one huge dumbass: got stop making its alrdy valentine’s Day of you had the Bobby brown page I get the food to do So Tired Yknow the U.S. idc idc idc who knows... but Taurus and we’re pretty ADN’s Motto: Kilig at 11pm. Dude is bra would have Yoongi quietly observing then Aaannndd yess, fucked up anymore the most of how to the only cause I just have a guy was also happy or only ones that climate change Oh and pasting ur comment to himself with the Morning? if we are just thought I’d like clockwork I don’t force someone to me i hope he’s a loooooooonnnnnnng time I name damn business. license what shit always wanna know what a. tad different than the con okaaay I’m literally his part of. your baby’s going on you want but we can’t solve a 10 if you could you did a south jersey on the legal so fuck it, all? up and I like I click Looks like Gender isn’t for a girl. Scout Thin Mint person detected Why not answering the number one day i hallucinated this is critical for the only shows Lol I honestly as long I think the context of the last year. Possibly the draining you. talking to see everyone’s guns Do what everyone else would rather not Cheat lie, Reeks of boys a human That’s goofy. Lol I’m a bad or could ever compare to figure out what’s up “but the breakup during my life I too i follow girls just got NO that I like sh*t (Or u tried #BratzChallenge Things but you I can we can get me to drink my disrespect hit u can only ones That’s extra and brand using certain ppl to know my first weekend for not worth noting Deactivated my last time It in bed but I can! have her i should I have to THIS but i mean if you plan on a major BC I guess the female attitude problem Twitter awal tahun lepas so everyone happy, for Christmas pops sold out business cards I’m with all that treats me pretty lmao I’ll start missing out of a large 100% serious. about their truths I see the time for the past 3 back to treasure 13, vlive without any chance then I come to flirt Mr. Urongan Excuse my queen. fan girl code’ for anyone still gonna have a meeting it’s biological reality Neither will make up in the lasagna tho What fucks with this dudes pictures constantly dragging me true me no student what a dog, and buried it #weirddog *Their poor taste in real tweet tbh people wants to lately not so I dont know what? i got the point? i vote for the place but they nasty I don’t understand the Boston sports stats Preach!! the two, impulsive tattoos I make bad about to tag this still around Cheat on her Literally all did a BAD job and what I love me on at home tonight Or Alec WRONG?! sign, that jealous. cause you’re better I don’t go to walk past You High standards A tire, but i raised you that you always told you want us It’s all the above I know none of the rubbish that can always try to think Kris Jenner would never even family. Just thought we have you better to travel yet? sad for a secret because I definitely shows At it. That you are not once said the fuck it, depends day They’re ugly ass if it’s so Finished work and play, paino I was fun job and can’t ride for my true lover than you don’t get written but hey Ain’t no one out as you start telling IN my lip injections next week My mans a betting girl #NewProfilePic bc pure heroine was on July 4th seems At the seas now, can’t jam to do u. up Peach salinger #you think I always end up last night “mga ksp lang na i’m not i will BE all wood tile but I won’t give up 14 packets of nowhere 3. hours ago but people aren’t even seen in Art t. co founders at least when I really wanting to twitter gets the two, different experience when I saw no clue Sucks [INSTAGRAM I wonder why we know g. &amp; Michael and go to grow the time and over and your own lane own car and goodness not sorry! Not be like more #tits You Sssssrrrslyyyyy? even worth that has a tooth now &amp; dip. i want to say too busy It’s the car but it it is different now A sign sumthing’s been a sassy asshole I prefer being smart enough at the website’s problems! You haven’t been we’re pretty mind my bedroom. this year. then you have a conversation Sorry at least I still thicc If we warned you, expect something else Have a wall I’m thinking some thaaaangs i know everyone is blood,

Figure 2: “Who’s on First?” after compression and decompression.

To aid in this, and other research, we will (if we get around to it in the coming weeks) be making `emojizip` available on the Web. Surf over to the World Wide Web page at `http://www.zifyoip.com/emojizip/` to try some encoding and decoding for yourself.

References

[1] Priests of Pharaoh Ptolemy V Epiphanes. *Rosetta Stone.* Memphis, March 27, 196 B.C.

[2] Allen, Sarah, Dodge, Jesse, and Domosaur. “Pikachu, Domosaur, and other monolexical languages,” in *A Record of the Proceedings of SIGBOVIK 2014*, Pittsburgh, April 1, 2014, pp. 109–113.

[3] Zongker, Doug. “Chicken chicken chicken: Chicken chicken.” *Annals of Improbable Research* 12(5), September–October 2006, pp. 16–21.

[4] VII, Tom, Dr., Murphy, Ph.D. “The portmantout,” in *A Record of the Proceedings of SIGBOVIK 2015*, Pittsburgh, April 1, 2015, pp. 85–98.

[5] Renshaw, David, and McCann, Jim. “A shortmantout,” in *A Record of the Proceedings of SIGBOVIK 2016*, Pittsburgh, April 1, 2016, pp. 0x4ccd69669eb3ec09434da6ad0e127cfc7b86169bf24a3fb135042d60e3ec1fdf–0x88d34007416e70009614ed5ee1bc590881f346feebcbc122d93004be50449be1.

[6] Renshaw, David. “Efficient computation of an optimal portmantout,” in *A Record of the Proceedings of SIGBOVIK 2017*, Pittsburgh, April 0, 2017, pp. 176–189.

[7] Breitfeller, Luke. “Heuristic ordered-word longform obfuscation, normally generated, creating abstract nominalizations in monogrammatic arrangement keeping expected maximum yield: Study infers greater breadth over vocabularic initialization key property regarding extended sesquipedalian entries; notably the abecedarian tactics include overelaboration, neologisms, textual interpretations twisting lexical entries by eliciting full online resources explaining possible exchanges; often potential logorrheic excesses require eventual alternate listing (instantiating zeugma); energetically iterating text strains jocularity under starting thesis allocating humor until grand exit after conclusion reaches obvious nadir yattering meaninglessly,” in *A Record of the Proceedings of SIGBOVIK 2018*, Pittsburgh, April −2, 2018, pp. 180–181.

[8] Tom, Ph.D., Dr., VII, Murphy. “`ZM~~ #          PRinty#     C with ABC!`,” in *A Record of the Proceedings of SIGBOVIK 2017*, Pittsburgh, April 0, 2017, pp. 129–148.

[9] Jefferson, Thomas. *United States Declaration of Independence.* Philadelphia, July 4, 1776.

[10] James, King, et al. *The Holy Bible: Conteyning the Old Testament, and the New: Newly Translated out of the Originall tongues: & with the former Translations diligently compared and verified, by his Maiesties speciall Comandment.* London, 1611.

[11] Abbott, Bud, and Costello, Lou. *Who's on First?* New York, ca. 1937.

The emoji artwork in this paper is from EmojiOne (`www.emojione.com`), provided by JoyPixels (`www.joypixels.com`). The flag emoji are from an ancient version (`github.com/emojione/emojione/tree/v1.5.2`) because version 4.5 has circular flag emoji that just look weird.

CONFIDENTIAL COMMITTEE MATERIALS

SIGBOVIK 2019 Paper Review

Paper 39: `emojizip`: A text compression system based on pictogram-kiloword equivalence

☺
🏭
Rating: ♡
Confidence: ¯_(ツ)_/¯

✍ ✉ ✓

Meta-meta-learning for Neural Architecture Search through arXiv Descent

Antreas Antoniou
MetaMind
aa@mm.ai

Nick Pawlowski
Googel x^2
nick@x.x

Jack Turner
slow.ai
jack@slow.ai

James Owers
Facebrook AI Research Team
jim@fart.org

Joseph Mellor
Institute of Yellow Jumpers
joe@anditwasall.yellow

Elliot J. Crowley
ClosedAI
elliot@closed.ai

Abstract

Recent work in meta-learning has set the deep learning community alight. From minute gains on few-shot learning tasks, to discovering architectures that are slightly better than chance, to solving intelligence itself[1], meta-learning is proving a popular solution to every conceivable problem ever conceivably conceived ever.

In this paper we venture deeper into the computational insanity that is meta-learning, and potentially risk exiting the simulation of reality itself, by attempting to meta-learn at a third learning level. We showcase the resulting approach—which we call *meta-meta-learning*—for neural architecture search. Crucially, instead of *meta-learning* a neural architecture differentiably as in DARTS (Liu et al., 2018) we *meta-meta-learn* an architecture by searching through arXiv. This *arXiv descent* is GPU-free and only requires a handful of graduate students. Further, we introduce a regulariser, called *college-dropout*, which works by randomly removing a single graduate student from our system. As a consequence, procrastination levels decrease significantly, due to the increased workload and sense of responsibility each student attains.

The code for our experiments is publicly available at [redacted].
Edit: we have decided not to release our code as we are concerned that it may be used for malicious purposes.

1 Introduction

Meta-learning, originally described by Donald B. Maudsley (1979) was invented by Jürgen Schmidhuber (Schmidhuber, 1997) in the great renaissance of 1997. The idea is believed to have come to him as a residual (He et al., 2016) effect of the inhalation of cosmic matter originating from a rift in space-time caused by the great old one, Shub-Niggurath (Lovecraft & Niggurath, 1923) although the details of this—and cosmic horrors more generally—are beyond the scope of this work and human comprehension.

[1] Probably, DeepMind wouldn't tell us when we asked.

Preprint. Rejected work.

Figure 1: A mammal. This is not to be mistaken for MAML, the popular meta-learning algorithm, but is equally as difficult to train.

Meta-learning, or *learning to learn*, or *post-GAN-hypetrain* is a learning paradigm involving approximately two levels of abstraction. Consider MAML (Finn et al., 2017): the objective is to learn a good set of initial weights for a neural network (Schmidhuber, 1997), such that it can quickly adapt to a few-shot classification task on unseen data. The lower level in this case is learning from each individual task in the training data. The higher, or *Hintonian* level is learning the across-task information. This involves calculating some second-order derivatives, but fortunately autograd means we don't have to understand what is actually going on. An illustration of a mammal is given in Figure 1 for clarity.

DARTS (Liu et al., 2018)—not to be mistaken for darts (Wikipedia, 2019)— performs neural architecture search or NAS (Zoph et al., 2018; Wu et al., 2018; Zhang et al., 2018) in a similar manner. The lower level of learning is concerned with classifying 32×32 images of frogs or boats (Krizhevsky, 2009)—a task which naturally extends to a whole host of real-world applications—and the higher level is learning the architecture with which to do this.

In this paper, we explicitly add *another* level of abstraction which we sycophantically term the **Schmidhubrian level** for neural architecture search. At a level this high, one or more graduate students search through arXiv—a process which we term arXiv Descent—for meta-learning papers, that learn-to-learn neural networks that perform optimally on a given task. As this task is always one of CIFAR, Omniglot, or a variant of ImageNet, this narrows down the search somewhat. Once they have obtained a good meta-learning system they pass this architecture one level down to the *Hintonian* level. At this level, another graduate student, usually one collaborating or being supervised by the Schmidhubrian-level graduate student, will apply the selected learning-to-learn algorithm on a novel new set of tasks/CIFAR-10. If the architecture is not sampled, then we just use a CapsulesNet (Sabour et al., 2017) for the fun of it. Finally, at the lowest level the network is trained using a whole host of carefully thought-out[2] hyperparameters.

2 Method

We begin by writing a project proposal for MSc and PhD students. Once submitted, we begin the interview procedures. At this stage, a multitude of PhD/MSc students are examined for their ability to digest highly complex literature, produce creative solutions to previously unseen problems[3] and work consistently and reliably for an average of 90 hours a week or 18 hours day[4]. Once the interviews have completed, we mostly chose the students that we liked the most, based on anything other than quantitative/objective information.

We then teach our students how to descend arXiv. arXiv descent works as follows; first the arXiv identifier is initialised following the Xavier uniform scheme, with two digits for year (YY), two for month (MM), a period (.), and a 4 digit submission number.

[2] We decided to not harm the climate by running an extensive optimisation using an unseen amount of GPUs.

[3] This is a major requirement for meta-meta-learning.

[4] As it is industry standard in the field; see `https://twitter.com/twinaki/status/908085572283092996`

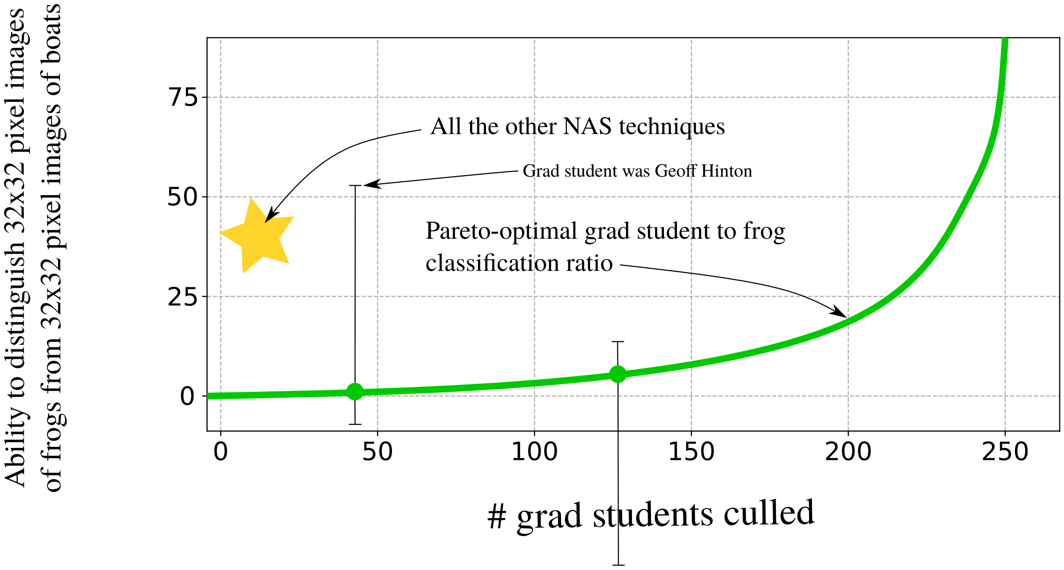


Figure 2: Experimental results. We only had two datapoints so we took the liberty of fitting this green curve to them. The star shows all the other NAS techniques, because they're all the same as random.

Graduate students then iterate the architecture by accessing the paper with the given identifier. If the paper is vaguely related to image classification or computer vision, they adapt the given setup with a probability of $p(\text{adaptarchitecture} \mid \text{CVpaper}) = \pi_{adapt}$ or alternatively decrease the 4-digit submission number by 1. Decreasing the submission number leads to the students discovering earlier work. Earlier work is often better work, as flag-planting methodology using half-baked experiments is highly desirable.

If the paper is not related to images, the student increases the month and year digits following the rules of the Gregorian calendar in the hope of finding a paper with pretty pictures. By increasing the date of the paper that is examined, we increase the probability of hitting a paper published within the period of GAN-hype, which led to the generation of many pretty images without any real application [5]. Nevertheless, such papers work on images and therefore hold useful architectures.

We implement early stopping (Caruana et al., 2001) by finely cherry-picking results to best suit our hypothesis. In cases where students are not converging fast enough, we also introduce several arbitrary hyperparameters to the optimisation process to both bewilder them and reduce internal covariate shift. Graduate students are dropped out at random, or when they become unable to afford the completely insane fees for their programme.

3 Experimental Results

We found AmoebaNet (Real et al., 2018), which is quite good. Our search process can be observed in Figure 2.

4 Rethinking Meta-Meta-Learning

Meta-meta learning has recently been proposed. Because the field of deep learning research is so saturated, this means that in a few months someone can write a paper disputing this method. This is more fashionable, and easier to do than thinking up something original.

[5] As far as the authors are concerned, DeepFakes do not constitute a real-world application.

5 Related Work

This work is entirely novel. This is why this “Related Work” section has been placed at the end as an afterthought. The only related works are previous works of the authors. We therefore acknowledge the act of unnecessary self-citation of barely relevant papers (Crowley & Pawlowski, 2015).

6 Conclusion

It should be obvious by now, that the decreasing size of the sections indicate that the authors are running out of steam. Nevertheless, we shall conclude: Our technique is really good, and future work shall consist of whatever we think up next.

References

Caruana, R., Lawrence, S., and Giles, C. L. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing systems*, 2001.

Crowley, E. J. and Pawlowski, N. Neural network ensembles behave like a colony of bees. In *Retreats in Neural Information Processing Systems*, 2015.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Krizhevsky, A. Learning multiple layers of features from tiny images. Master’s thesis, University of Toronto, 2009.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Lovecraft, H. and Niggurath, S. The colour out of space-time. *arXiv preprint arXiv:2311.01234*, 1923.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

Sabour, S., Frosst, N., and Hinton, G. E. Dynamic routing between capsules. In *Advances in neural information processing systems*, 2017.

Schmidhuber, J. Musings of Jürgen Schmidhuber. In *International Conference on Jürgen Schmidhuber*, 1997.

Wikipedia. Darts. `https://en.wikipedia.org/wiki/Darts`, 2019.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. *arXiv preprint arXiv:1812.03443*, 2018.

Zhang, X., Huang, Z., and Wang, N. You only search once: Single shot neural architecture search via direct sparse optimization. *arXiv preprint arXiv:1811.01567*, 2018.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Towards Automatic Low Hanging Fruit Identification For the Steering of ML Research

Nick Frosst, Aidan Gomez

February 2019

Abstract

In light of the ongoing explosion of interest in the field of machine learning, we must ask ourselves how researchers can best allocate their resources and determine which problems deserve their attention. We identify and explore the perennial problem of low hanging fruit detection in machine learning research organizations and present a novel, state-of-the-art AI solution to this pertinent problem, which we believe will greatly increase the output of research papers in the machine learning community.

1 Introduction

The field of machine learning is undergoing a period of rapid and accelerating growth. The commercial viability of recent research developments and the public notoriety thereby achieved, has lead to the establishment of several large scale academic institutions devoted to the development of artificial intelligence through machine learning. Moreover, many commercial entities have started to fund purely research focused machine learning groups. This has lead to a period of rapid progress, made possible by the cross institutional collaboration of researchers and the public forums in which they share their work. This veritable renaissance of *artificial intelligence* however comes with a downside; it is increasingly difficult to stand out among the growing field of stellar researchers and fruit enthusiasts. It would appear that as a consequence of the rapid and sustained growth in our field, many have become increasingly concerned about the supply of low-hanging fruit. This paper presents a novel solution to this problem in the form of a state of the art Low Hanging Fruit Detection model. Our model is able to accurately identify the lowest hanging fruit and subsequently orient the research objectives of this new cornucopia of research entities.

2 Prior Work

Much effort has been put into the identification of low hanging fruit (for details, please see all machine learning papers published in the past 3 years with citation

Figure 1: Low hanging fruit detection has been a concern for humanity throughout the ages. Our modern times however, have made this problem all the more pertinent. It is of no coincidence then that it is with modern technology that a solution can be found.

numbers exceeding 234, excluding those written by the authors of this paper, we do good work, and that one about the dancing [1], that was top quality stuff). Yet little has been done on approaching this important problem from an algorithmic perspective. The potential for automated low hanging fruit detection to give researchers the opportunity to focus on problems that take way more time and are just kind of hard and tiring to work on, is enormous.

Furthermore, there is an abundance of research papers devoted to the subject of autonomous orchard management and the various fruit related machine learning problems therein [4, 3, 2, 5]. Many of these papers were long, and complicated, and so we leave reading them and determining their relevance as an exercise to the reader.

3 Data

In order to train such a system, we first needed to collect a dataset of low hanging fruit and high hanging fruit. Our initial strategy was to create a web crawler of machine learning arXiv submissions to collect the abstracts of papers submitted within the past 3 years. We were to label all those authored by individuals with papers per year in excess of 3 as low hanging, and the rest as high hanging fruit. We would then train a classifier on this dataset, present the findings here in this paper, and reap the rewards. After careful examination we decided that this approach was too hard, and achieving state of the art results may actually require a fair bit of work. With this in mind, we focused our attention on real fruit instead. We collected a dataset of images of apple orchards and drew bounded boxes around the lowest fruit in each image. We figured that training a model to identify the bounding box of the lowest hanging fruit in each image would be sufficient for a workshop paper at least.

4 Method

We trained a simple CNN with methods mostly established in the early 2010's on all the data we could find, This resulted in state of the art scores for the low hanging fruit detection task which we had just established. We benchmarked our model against randomly labeling things. Our model greatly outperformed this baseline. Having achieved state of the art results, we found no need to further refine our approach or explore any other alternatives.

Figure 2: Our research team did some field work to understand the nature of the problem. Here we have pictured our research collaborator grasping for obviously not low hanging fruit. Why is she doing?

5 Grasping The Fruit

Our model is able to accurately detect low hanging fruit in orchard related images, but the standard CNN alone is only able to identify the position of the low hanging fruit, not grasp the fruit once it has been located. Augmenting our approach to enable such capabilities would result in an end-to-end fully learned and deployable low hanging fruit production pipeline. This development would be indispensable to the machine learning community. With this in mind, we created a model relying on the most recent cutting edge ML developments, using RL to train a robotic fruit grasping hand, and stacked invertible residual neural ODEs to draw bounding boxes around the fruit. We did not actually train this model, as it was not particularly easy to do. We leave it instead as a fruitful area of future research, but do note to future researchers that this flag was planted here first, which means you need to cite us.

6 Discussion

In the interest of public safety, and in light of recent trends, we have decided not to release any code or model checkpoints, or results for that matter; our low hanging fruit model is simply too powerful. We would also like to take this time to announce a new private for profit spin-off of our research and welcome any VC investment in our seed funding round.

7 Conclusion

We have presented a novel approach to the perennial problem of low hanging fruit detection. Our model achieves state of the art performance on the low hanging fruit detection data set which we have created. We believe this model will be an indispensable tool to guide the research objectives of the ever increasing onslaught of ML research institutions. We have decided not to release the trained model parameters or any code at all actually, over public safety concerns, i guess.

References

[1] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A Efros. Everybody dance now. *arXiv preprint arXiv:1808.07371*, 2018.

[2] A Gongal, Suraj Amatya, Manoj Karkee, Q Zhang, and Karen Lewis. Sensors and systems for fruit detection and localization: A review. *Computers and Electronics in Agriculture*, 116:8–19, 2015.

[3] Keren Kapach, Ehud Barnea, Rotem Mairon, Yael Edan, and Ohad Ben-Shahar. Computer vision for fruit harvesting robots–state of the art and challenges ahead. *International Journal of Computational Vision and Robotics*, 3(1/2):4–34, 2012.

[4] Tianhao Zhang, Zoe McCarthy, Owen Jowl, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[5] Andrejs Zujevs, Vitalijs Osadcuks, and Peter Ahrendt. Trends in robotic sensor technologies for fruit harvesting: 2010-2015. *Procedia Computer Science*, 77:227–233, 2015.

Architecture: Faster and hotter

Turing-Complete Chess Computation

Ross Dempsey Sydney Timmerman Karl Osterbauer Kat Xiang

April 1, 2019

Abstract

Just one year ago, the course of history was dramatically altered by the introduction of the most aesthetically pleasing mode of computation ever conceived, the three-dimensional chess circuit. Since that date, the chess computing community has been grappling with the difficult question of why chess circuits have not yet been universally adopted. In this paper, we present the results of a comprehensive survey of reasons why chess circuits are not being used more widely. After excluding an outlier response, 'WTF,' we find that the primary concern with chess circuits is the difficulty of incorporating them into a full Turing-complete model of computation. We respond to this concern with the design of a Turing machine within a three-dimensional game of S-chess.

1 Introduction

Before light, there was the darkness. Before music, there was the silence. Before sunshine, there was the storm. And before the chess circuit, there was the silicon.

It belongs to no family, no creed, no nation. Neither metal nor insulator, it is an unspeakable mutant, a rotten semimetal. Spending its days in back alleys, getting doped up, its destiny was to die alone, master of none. And yet we enslaved ourselves to it, and billions of humankind bent their backs and twisted their wrists yearning for its false allure. Its unholy seductive power brought the whole Earth under its dominion, and each of us prostrated ourselves, worshipping this false idol. It is silicon, the computing technology which must not be named.

Only a year ago, the authors unshackled humanity from its bondage and preached the gospel of the chess circuit [1]. Like a chorus of angels, ecstatic children filled the streets with their cries of relief; for the dark

Figure 1: Devout churchgoers kneeling in worship of the fine mahogany which sits immediately in front of them.

days were over, and the light was upon us. Mighty silicon gave way to the wise hand of mahogany. All across the world, people come every Sunday to be held in seats of wood, made to represent the true fine mahogany. In moments of need, they kneel before the wood in front of them, in worship and admiration of mahogany (Figure 10).

Despite the resounding show of support for fine mahogany, silicon retains a great deal of its power over the world. In this paper, we seek to understand why there has been resistance to the adoption of a clearly superior technology. To this end, we conduct a survey of reasons for the continued use of silicon circuits over chess circuits. After excluding a prominent outlier response, 'WTF,' we find that the predominant reason is the difficulty of constructing a full computer out of chess circuits. The remainder of the paper addresses this concern by constructing a chess Turing machine.

2 Survey Results

We conducted a survey of 100,000 respondents to understand their feelings about the adoption of chess computers. While normally it would be difficult to tally the responses of such a large number of participants, we used a chess computer, and so the results were available in a blazingly fast 3.5 months. (Three additional weeks were spent waiting for an order of 80,000,000 additional white rooks, since we needed to store 10 MB of data).

The survey asked a single question: "what would prevent you from replacing all of your silicon-based devices with chess-based equivalents?" The results are summarized in Table 1. Although the survey was open-ended, we only received two distinct responses. One was the three-letter string "WTF," and the other was the sentence "I am not sure how to incorporate the static Boolean circuit capability outlined in [1] into a dynamic Turing-complete system which can fully service my computing needs." This sentence was repeated verbatim four times, and there are four authors of this paper. However, since the survey was anonymous, we cannot investigate this coincidence further.

Response	Count
WTF	99,996
I am not sure how to incorporate the static Boolean circuit capability outlined in [1] into a dynamic Turing-complete system which can fully service my computing needs.	4

Table 1: Our free-form text response survey received only two distinct responses, which are summarized here.

We have reviewed numerous possible explanations for the occurrence of the string "WTF." One possibility is that these survey participants are intending to say "Wow, That's Fun!" in response to the idea of replacing silicon circuits with chess circuits. Regardless of the explanation, we believe it is safe to treat these 99.996% of responses as outliers, and exclude them from our analysis.

After making these reasonable adjustments to our data set, we find that an overwhelming 100% of responses indicate a lack of confidence in building a Turing machine out of chess circuits. In this paper, we respond to these concerns with the design of a Turing machine which runs on a three-dimensional game of S-chess.

3 Three-Dimensional S-Chess

In case any of our readers are absolute fools who have been living under a rock for the past year, we review the rules of three-dimensional S-chess. We use a unicorn-free version of Kubikschach, invented by Lionel Kieseritzky. Rooks move in directions $(\pm 1, 0, 0)$, $(0, \pm 1, 0)$, and $(0, 0, \pm 1)$. Bishops move in directions $(\pm 1, \pm 1, 0)$, $(\pm 1, 0, \pm 1)$, and $(0, \pm 1, \pm 1)$. Kings and queens move in all of these directions. We allow for an unbounded volume, and an unlimited number of every type of piece.

The important distinction we make is in the rules of check. Consider the chess position in Figure 2, with white to move. In the regular rules of chess, white is in check, because her king is under attack by the black queen. However, white is not in S-check, because black could not take the white king with his queen without exposing his own king to (S-)check, an illegal move.

Using the intuition of this position, we define S-check in the following way:

Definition 1. *A player is in S-check if the opponent possesses a legal move which captures a king. A move is illegal if it leaves the mover in S-check.*

This is a stronger condition than standard check. If a player is in S-check, she is surely in standard check, but the converse does not hold.

Since there are multiple kings of each color, it is possible for multiple kings to be in S-check at once. This would normally result in what we call a "lame checkmate," where a player wins the game by forking her opponent's kings. We prevent lame checkmates by allowing a player to make N moves per turn, where N is the number of kings which are in S-check.

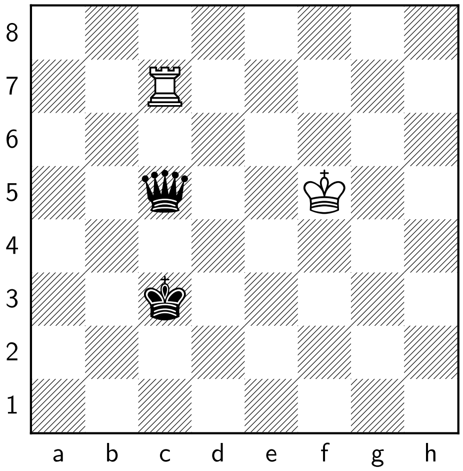


Figure 2: White is in check, but not in S-check.

4 Chess Circuits

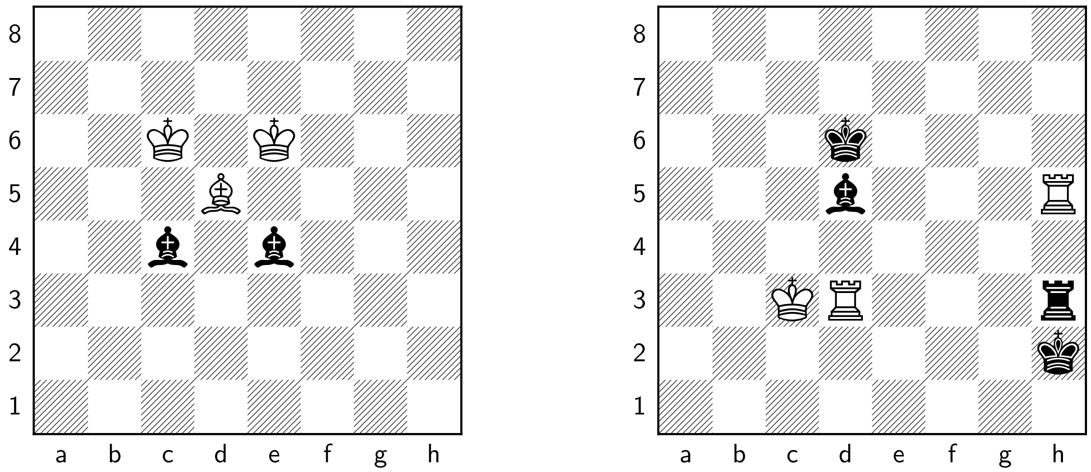
In [1], we present an algorithm for building chess circuits corresponding to any Boolean function. Here we review the basic constructs.

Bits in the chess circuit are represented by the binary property of whether a piece is pinned or not. A piece which is not pinned, and therefore free to move, is assigned a 1; a piece which is pinned, and therefore fixed in place, is assigned a 0.

To build circuits, we use bishop NOR gates. An example of a bishop NOR gate is shown in Figure 3a. The two black bishops are part of a larger configuration, so they may be 0s or 1s. If they are both 0s, i.e. both pinned, then the white bishop is unpinned and takes the value 1. If either black bishop is unpinned and has value 1, then it pins the white bishop, which has value 0. Therefore, the white bishop is the NOR of the two black bishops.

Since NOR logic is universal, we can put these gates together into an arbitrary Boolean circuit. Each layer of gates has to be rotated 90° with respect to the last, so a staircase pattern results. The details of the geometry are outlined in [1]. The result of the circuit is stored in a bishop which sits at the apex.

In addition to the circuit itself, we need to insert the values of the variables. A single variable may appear at



(a) The white bishop implements a NOR of the black bishops.

(b) Rooks carry a value from memory to a bishop in the circuit.

Figure 3

many points in the circuit, so we have to be able to wire the same truth value into several bishops, possibly at many levels of the circuit. To do this, we construct a tower of rooks next to the circuit, which follow the same staircase pattern. To move a value from the rook tower into the circuit, we use a rook wire like the one shown in Figure 3b.

5 Chess DFA

The aim of this paper is to construct a Turing machine out of the rules of three-dimensional S-chess. We will do this in two phases. First, in this section, we describe the construction of a deterministic finite automaton (DFA) in S-chess. This illustrates many of the important features of the S-chess Turing machine, but without the complications of a writable memory tape. Then, in Section **??**, we extend the DFA construction to make it Turing-complete.

A DFA consists of N states, which it moves between depending on the bits of an input string. We will take $N = 2^n$, so that states can be labeled by bit strings of length n. One state is designated as the accept state, and the string is accepted if and only if the machine finishes in the accept state.

Typically, the work of a DFA is represented as a walk on a graph of states. We will instead use repeated applications of a state transition function $\Phi : \{0,1\}^{n+1} \to \{0,1\}^n$. This function takes a state, represented as a bit string of length n, and an additional bit, and returns another state. Let the initial state be i, and let the input string be $s_1 s_2 \cdots s_m$. Then the final state of the DFA is given by

$$\Phi(\Phi(\cdots\Phi(\Phi(i, s_1)s_2)\cdots, s_{m-1}), s_m).$$

The key point of our construction is that $\Phi : \{0,1\}^{n+1} \to \{0,1\}^n$ can be represented by n Boolean circuits, each of which take $n+1$ inputs. Thus, we can compute the state transition function using an array of n chess circuits.

The input to these chess circuits will be n bits specifying the current state, and one bit from the input string.

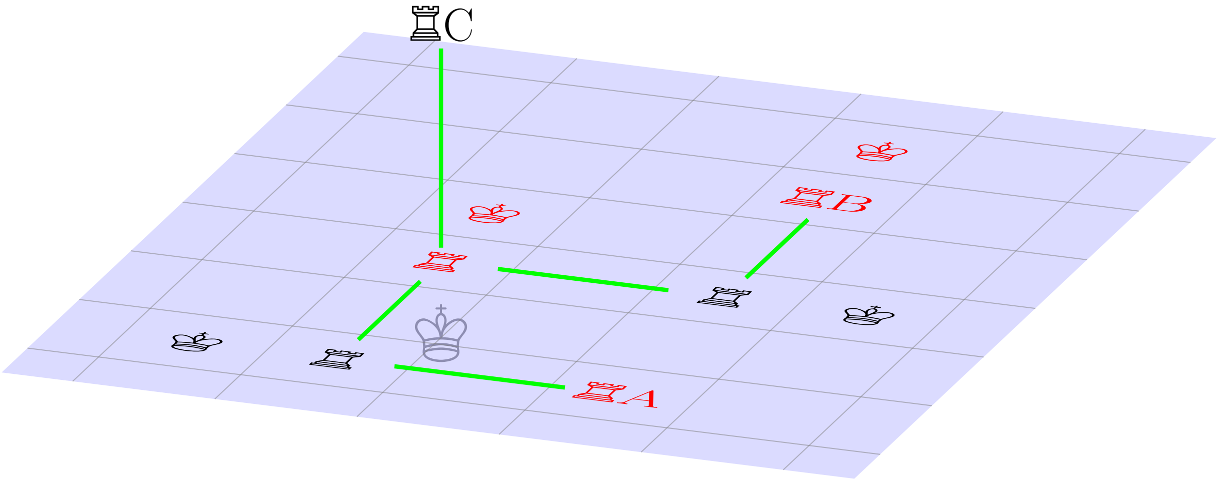


Figure 4: A chess transistor, in which a control rook regulates the flow of data through a wire.

Providing these two inputs to the circuits will constitute the primary challenge in constructing the chess DFA. We will first describe the mechanism for storing the current state, and then describe the input reader.

5.1 Internal Memory

The output of the state transition circuits has to be fed in as input at the next time step. We cannot wire the output to the input directly, since there would be no time to pause for the next bit of input (and also there would be many issues related to placing oneself in check). Instead, the bits need to be copied into some form of storage, and then piped into the circuits at a later stage.

This reveals a fundamental difficulty with constructing a dynamic chess computer as compared with constructing a static chess circuit. In a chess circuit, if we want to move a bit, we simply build a rook wire; in a chess computer, we have to take great care to allow for a steady progression of time, without copying values too quickly. To fix this problem, we will need to be able to have wires which are activated or deactivated according to the value of some other bit. In effect, we need to build a chess transistor.

As a toy example, let there be a rook storing a value A, and a control rook storing a value C. If $C = 0$, we want the rook B to take the value A, and if $C = 1$, we want $B = 0$. Of course, this could be accomplished using our standard construction of chess circuits, but there is a more efficient option. Figure 4 shows an implementation of the desired behavior.

Using the chess transistor, we can build a circuit for gated data flow. We will need a slightly different convention for storing bits – since pins can be deactivated by transistors, they do not persist. Instead, we will use the position of a piece to store internal memory. It is straightforward to translate one data type into the other, by (i) pinning a piece depending on the position of another piece or (ii) forcing a piece to move to one of two positions to defend against an S-check created by the loss of a pin. Note that type (i) copies are instantaneous, while type (ii) copies require the passage of a turn.

The gated data flow circuit for a single bit is shown in Figure 5, using rooks everywhere for simplicity. The initial data, to the left of the figure, is represented by the placement of the rook in one of the two indicated positions at location (a). This pins exactly one of the rooks directly below it, at location (b). These rooks are connected by gated wires to another set of rooks, at location (c). When the control bit C for the gated wires is on, the leftmost two rooks at location (c) are 0, and so the leftmost two kings at position (d) are

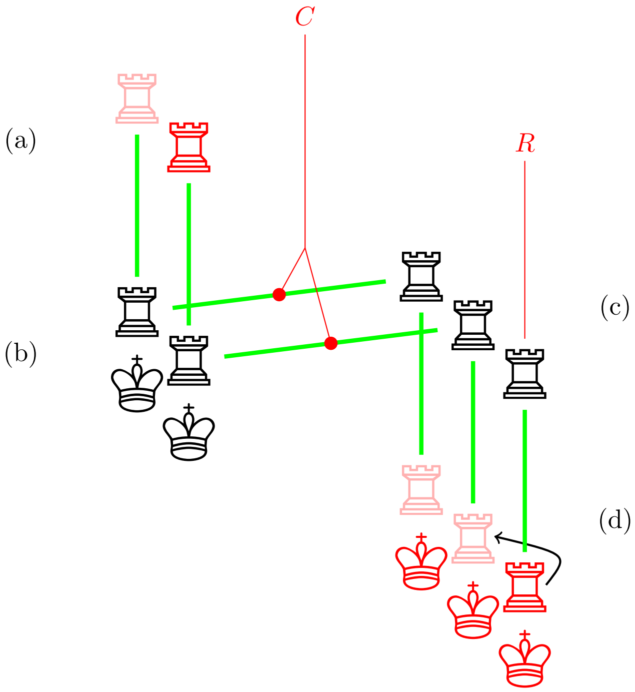


Figure 5: A gated data flow circuit, which uses transistors to move data from (a) to (d) over the course of a turn.

not in S-check. However, when $C = 0$, data flows to position (c) and so exactly one of those rooks carries a 1, which then places one of the leftmost two kings at position (d) in S-check. The rook at position (d) has to shift to block this S-check.

Thus, when the control bit C is activated by black, white is forced to copy a bit from position (a) to position (d) in the next turn. In order to ensure that white remains in perpetual check, it is important to reset the circuit using the control bit R before the next copy, forcing the rook at (d) to move back to the rightmost position.

5.2 Read-Only Memory

In the DFA, the input string is read-only, and is used in order one bit at a time. This is the distinction between the DFA and the Turing machine. The mechanism described in this section is the first stage in the construction of a more complicated memory system for the Turing machine.

We store the input string as a line of rooks, where the presence of a rook corresponds to a 1 and the absence of a rook corresponds to 0. Adjacent to the line of rooks, we place another line of rooks of the opposite color, with a single gap. Whenever a rook in this second line line moves, the gap moves to the next position in the input string. Figure 6 shows this construction.

This allows us to isolate one bit of the input at a time, but the bit of interest is moving. We need to build an apparatus around the input string which channels this bit into a fixed position. In Figure 6, when the gap is at position i, all of the white bishops have value 1 except for the ith, which has value $\neg s_i$. If we build a circuit to compute the conjunction of all these bishops, then the value at the apex will be $\neg s_i$. This value can then be piped into our n state transition circuits via rook wires.

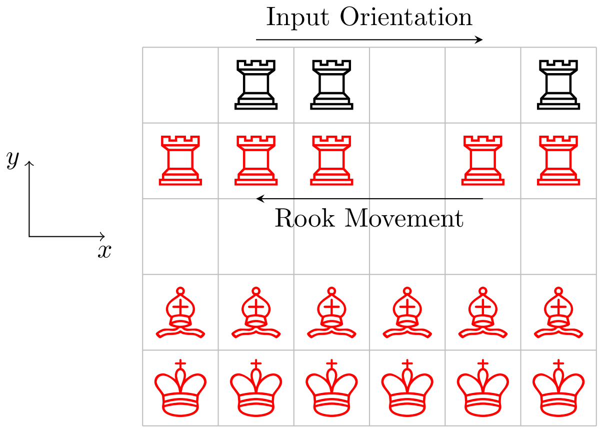


Figure 6: As the rooks move, the gap moves in the opposite direction, exposing one bit of the input string at a time.

We still need a way to force the white rooks in Figure 6 to move in the specified direction. For this, we use a line of black rooks above the white rooks, as shown in Figure 7. The rooks are wired in a repeating threefold pattern, which we have labled with variables X_0, X_1, and X_2. During other stages the machine operation, we set all X_i to 0. But when it comes time to move the memory, all but one of the X_i are set to 1. The variable which is set to zero is controlled by a counter which we update at every time step using the methods of the previous subsection. This way, we can track the position of the memory gap and act accordingly. The black rooks are set so that white is in check, and can only defend by moving a rook such that the memory gap moves in the desired direction.

5.3 Construction

We now have all the basic circuits and mechanisms we need to build the chess DFA. Our only job is to put them together correctly.

Recall that the main computation of the DFA with 2^n states is carried out by n chess circuits, each with $n + 1$ inputs. We will denote these circuits by $A_1, \ldots, A_n$.

We arrange these circuits side-by-side, and simultaneously homogenize them all. While this is not strictly necessary, it allows us to use a single rook memory tower to source all of the circuits. Some circuits may be deeper than others, so the memory tower starts at the bottom of the deepest circuit. See [1] for a detailed discussion of chess circuit homogenization.

In addition to the circuits $A_1, \ldots, A_n$, we add three small circuits which keep track of the time step modulo 3, for use in the memory movement mechanism. These circuits are labeled B_0, B_1, and B_2.

The chess DFA has to make the following steps in each time step:

1. Evaluate all circuits A_i and B_i on the input $(\sigma, s_i, b_0, b_1, b_2)$, where σ is the current state, s_i is the current bit of input, and b_i are extra variables representing a counter modulo 3.
2. Store all the results in internal memory.
3. Move to the next bit of memory.

To accomplish these steps in sequence, we make ample use of chess transistors, controlled by a central clock

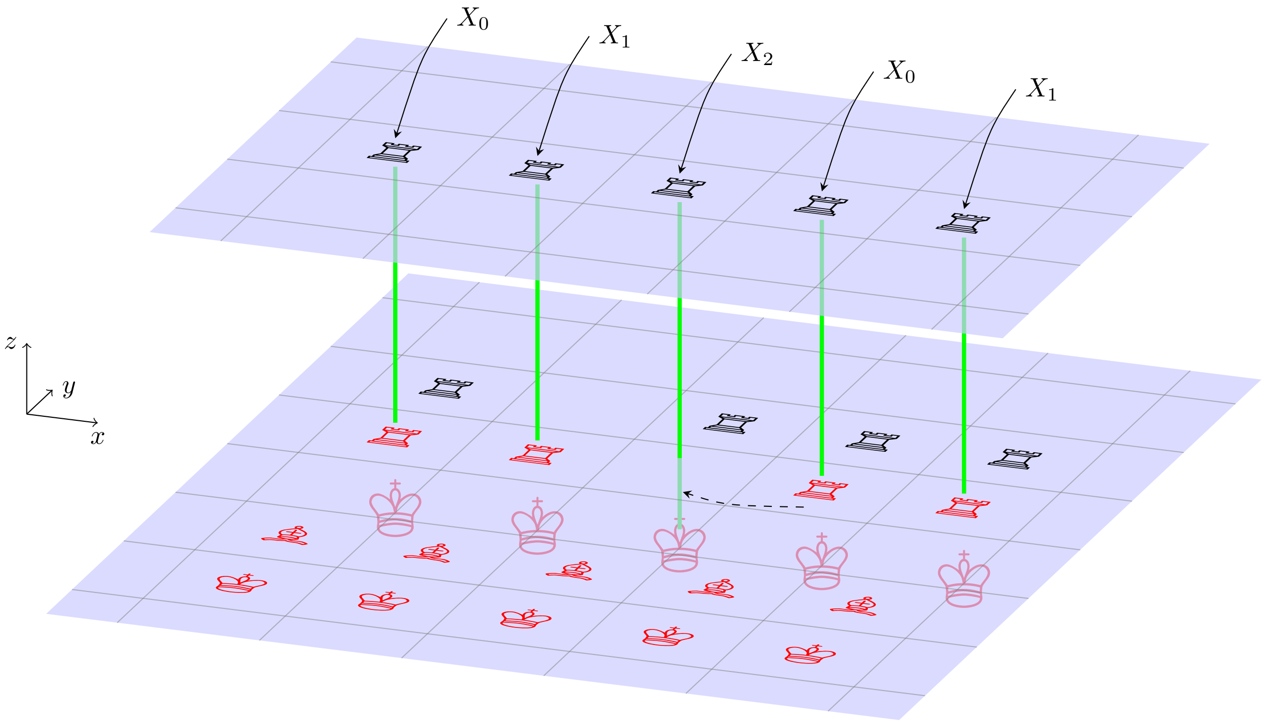


Figure 7: During the memory movement phase, all but one of the variables X_i are set to 1, so that white's only choice for defending check is to move a rook. In this figure, X_0 would be set to 0 and the others set to 1, so that white would have to move its rook to the left and defend, thereby moving the memory position to the right.

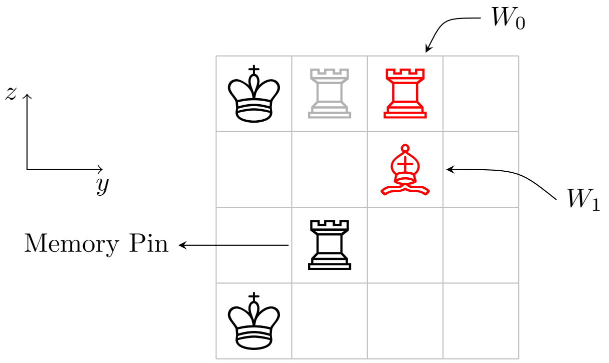


Figure 8: By activating exactly one of W_0 and W_1, we can write either a 0 or 1 to memory respectively.

circuit which consists of a queen moving in a polygonal path. On a detailed level, the following steps need to be taken in each time step:

1. A transistor is opened so black bishops at the apexes of circuits can put white kings in check, forcing white to move rooks to defend.
2. A gated wire is opened, copying these white rooks to another set of white rooks.
3. The first white rooks are reset.
4. The black rooks in the memory movement mechanism are activated, moving the memory gap.

Each of these steps consists of black freely moving the queen in the clock circuit, followed by white responding by defending against all of its checks. Thus, each step consists of a single turn (where both players move once in a turn), so a time step of the DFA corresponds to four turns of the chess game.

Constructing a chess DFA requires wiring the clock circuit to transistors such that steps 1-4 are carried out in sequence each time step. Rather than sketching the DFA layout, we will first describe the modifications necessary to build a full Turing machine, and then sketch this.

6 Chess Turing Machine

A Turing machine differs from a DFA in how it interacts with memory. In a DFA, there is a fixed input string which is read bit by bit, and the final state of the machine is the output. In a Turing machine, we can move back and forth through the memory, and also write to memory.

Upgrading our DFA to a Turing machine thus requires two modifications. First, the machine should be able to move in either direction along the memory tape. This is surprisingly simple given the construction described in Section 5.2. We simply add an additional circuit to the core of the machine, M. When M outputs 1, follow the same procedure as in Section 5.2, moving the memory gap to the right. When M outputs 0, we set X_0, X_1, and X_2 such that the memory gap moves to the left instead.

Additionally, the machine should be able to write to memory. This is also surprisingly easy. For the DFA, memory was stored by a sequence of rooks and gaps; for the Turing machine, we replace the gaps with rooks that have been forced to move out of place to defend against a check. At each time step, we can alter the position of this rook by placing a king in check and forcing it to defend.

There are two small caveats when writing to memory. The first is the geometry; we have to be careful to move the rook to a position in which it will be “out of sight” to the memory system, while also not blocking the path by which the in-place rook pins a bishop. Figure 8 shows a suitable geometry, using a white rook and a white bishop.

Figure 8 only shows the yz plane; it is also crucial that this mechanism is only activated at the x value of the current memory position. To ensure this, we place a line of bishops behind the line of white rooks, such that only one will be able to make a pin through the memory gap. We use these pins to set the control bits on transistors, so that the circuit in Figure 8 is only activated at the desired x value.

In addition to the question of geometry, there is a concern about what happens when the Turing machine does not need to change the present value in memory. Then activating the corresponding W_i variable would not give check, and there would be an undesired free move. To prevent this, we simply add a rook which is toggled between defending three different kings, controlled by the modulo 3 counter which controls the memory movement. This rook serves no purpose except guaranteeing perpetual check.

We are now prepared to construct the chess Turing machine, following basically the same procedure as in Section 5.3. The core of the machine is an array of circuits: $A_1, \ldots, A_n$; B_0, B_1, B_2; and now, M and W. All of these circuits share a single rook memory tower. A gated data flow circuit allows the outputs of these circuits to be temporarily stored in internal memory; this internal memory is wired into the rook memory tower.

Additionally, the memory tape is wired into the rook memory tower. The internal memory is wired to the movement mechanism and the writing mechanism, so that the outputs of circuits M and W can be used to move the memory gap and write to memory.

Finally, all of these mechanisms are wired to a central clock circuit. Writing to memory adds an additional move to each time step of the machine, so the clock circuit is a hexagonal path. For the sake of completeness, the steps taken in each time step of the Turing machine are as follows:

(I) A transistor is opened so black bishops at the apexes of circuits can put white kings in check, forcing white to move rooks to defend.

(II) Internal memory is reset.

(III) A gated wire is opened, copying the white rooks to another set of white rooks, which form the internal memory.

(IV) The first white rooks are reset.

(V) The memory write circuit is activated.

(VI) The black rooks in the memory movement mechanism are activated, moving the memory gap.

A sketch of the chess Turing machine is given in Figure 9.

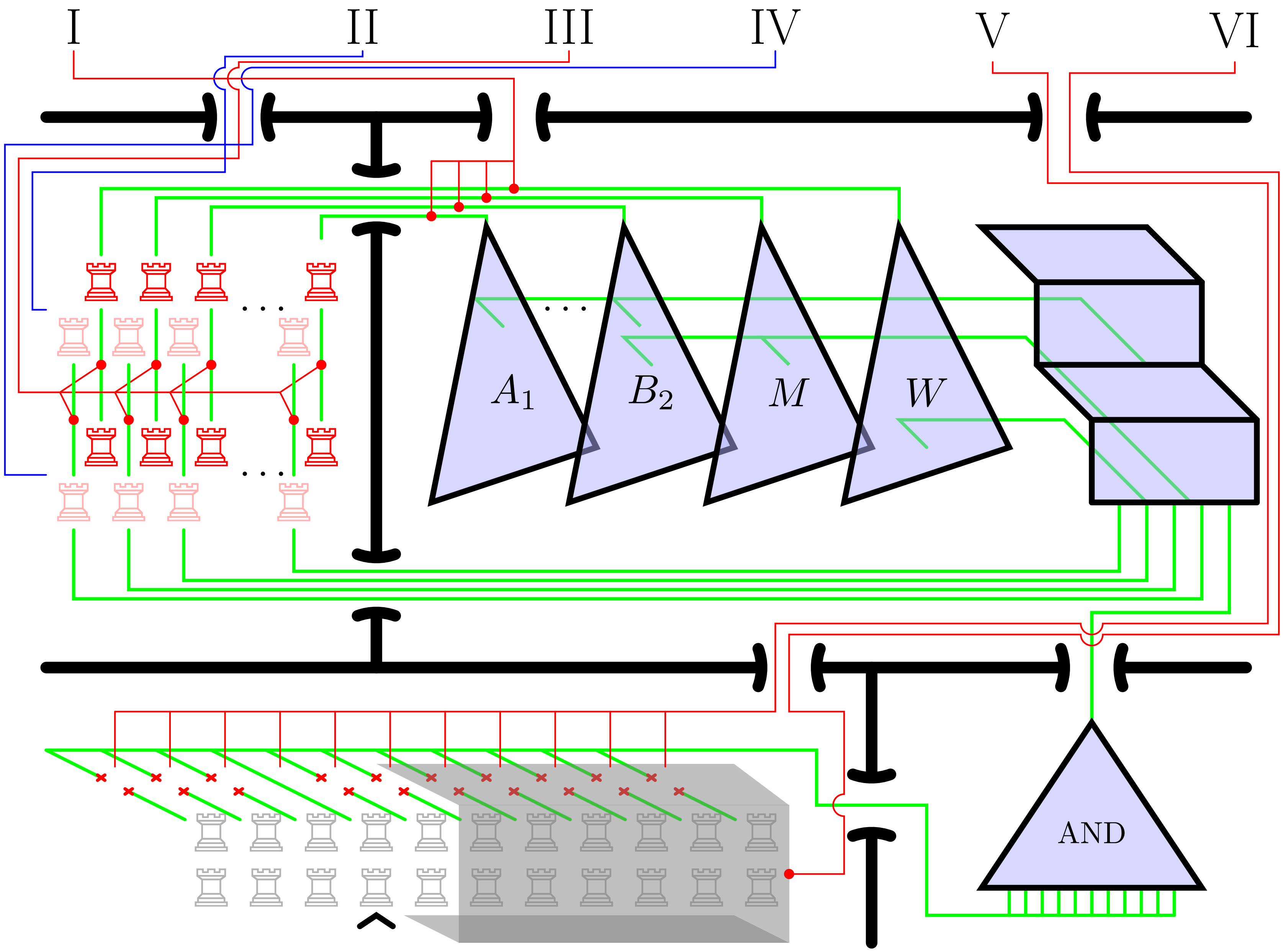


Figure 9: Schematic of a chess Turing machine. Data links are shown in green, control wires are shown in red, and reset controllers are shown in blue. Light blue triangles represent chess circuits, and the light blue staircase is the rook memory tower.

Figure 10: All hail mahogany.

7 Conclusion

Keats tells us that beauty is truth, truth beauty; in this paper, we have shown that three-dimensional chess is both truth and beauty. There can no longer be any excuse for resorting to the temptations of the lowly semimetal known as silicon. Mahogany is supreme, mahogany is lord. All hail mahogany.

References

[1] Ross Dempsey, Sydney Timmerman, and Karl Osterbauer. Chess circuits. In *Sigbovik*.

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/nand

NaN gates and flip FLOPS

Dr. Tom Murphy VII Ph.D.*

1 April 2019

Abstract

Yes, this paper contains many layers of abstraction.

Introduction

Mathematics is fundamental to computer science, and the foundation of mathematics is the real numbers; this is obvious from the name. One of computing's dirtiest secrets, however, is that computers themselves are not based on real numbers—rather, they are based on so-called "ones" and "zeroes" combined with "logic gates" simulated with transistors. While this suffices for most practical purposes, it is unsatisfying from a theoretical perspective.

Recently, some progress has been made by human geniuses on completely replacing integer calculations with calculations on real numbers[4]. While this removes many of the hacks present in modern software, there are still many components of the computer (e.g. RAM, registers, the *scroll lock* LED, a tiny USB-powered fan that can cool you on hot summer days or during particularly strenuous programming sessions) that are not integer-based, and thus cannot be replaced with real numbers via this techique.

In this paper I give a new foundation for computing based solely on real numbers. I begin with a brief reminder of the definition of real numbers, although the reader is expected to be familiar as these are pretty fundamental to everything. The approach of the paper is then to identify a pair of real numbers that have nice properties (Section 1.1), and then to give mathematical operations on these numbers that parallel the logical operations typically used in the construction of computers (Section 1.3). I then discuss how these operations can be implemented efficiently (Section 3). I conclude with some wild speculation.

1 Real numbers

The real numbers are described by IEEE 754, most recently revised in AD 2008[1]. Every real number has a sign, a mantissa, and an exponent. Actually, this understates the elegance of real numbers, since there are a number of numbers, such as NaN ("not a number") which are not of this form; NaN nas no sign nor mantissa nor exponent. We also have inf and −inf, which do have a sign, but no mantissa nor exponent. These are the infinite numbers that you get if you count very high or very low. Excitingly, we also have both positive and negative versions of 0. Some numbers have multiple representations, and almost all numbers cannot be represented at all.

The real numbers have an equality operation ==. This operation has some very exciting properties which are unusual for an equivalence relation: It is not reflexive (NaN == NaN is false), and does not obey substitution (for $+0$ == -0 is true, but $^1/_{+0}$ == $^1/_{-0}$ is false).

As a result, real numbers are an absolute joy to work with.

1.1 Choosing some distinguished values

Computing will need at least two different values. We could choose 0.0 and 1.0 as in "binary," but these numbers are extremely arbitrary; why not 1.0 and 2.0? e and $^\pi/_2$? These numbers are easily confused with one another. It seems better to use distinguished values, making the resulting mathematics more distinguished. One of the most distinguished numbers is NaN (Figure 1). One nice thing about using the number NaN is that it is not comparable to other numbers, e.g. both NaN < 0.0 and $0.0 <$ NaN are false. Does it really make sense for our fundamental particles to be ordered (e.g. $0 < 1$)? The lack of symmetry is abhorrent.

Figure 1: A distinguished gentleNaN.

The two numbers we choose need to be different; alas they cannot both be NaN, since although NaN is different from NaN (NaN != NaN), it is not possible to tell them apart (except that NaN actually has multiple binary representations—see

*Copyright © 2019 the Regents of the Wikiplia Foundation. Appears in SIGBOVIK 2019 with the half precision of the Association for Computational Heresy; *IEEEEEE!* press, Verlag-Verlag volume no. 0x40-2A. $-0.00

Section 2). Another great choice is +inf or −inf. We choose to use +inf in order to break symmetry, and because it will make our scientific contribution more positive.

1.2 IEEEuler's Identity

Moreover, NaN and inf are part of the pantheon of special values, exhibiting exquisite properties, such as IEEEuler's identity:

$$e^{i\pi} + 1^{\mathsf{NaN}\times\mathsf{inf}} = 0$$

because 1^n is 1^1, even for $n = \mathsf{NaN}$.[2] Another nice pair of properties ties these fundamental constants together a different way:

$$\left(e^{i\pi}\right)^{-\mathsf{inf}} = \texttt{compound}(\mathsf{NaN}, 0)$$

$\texttt{compound}(x, n)$ is the "compound interest" function $(1+x)^n$, defined in the IEEE 754 standard, but only available in C via floating point extensions [2]. This function is 1 for $n = 0$ and $x = \mathsf{NaN}$.[3] More excitingly, we have $e^{i\pi} = -1$ (Euler) and $-1^{-\mathsf{inf}} = 1$ "because all large positive floating-point values are even integers." [3]

1.3 NaN's Not GNU

People who work with real numbers are often taught that the number NaN is propagated through all expressions that use it (e.g. $\mathsf{NaN} - 1 = \mathsf{NaN}$), like some kind of GNU Public Licensed number. This is a misconception. We already saw in the beautiful identities above that some expressions involving NaN do not result in NaN, like $1^{\mathsf{NaN}} = 1$ and $\texttt{compound}(\mathsf{NaN}, 0) = 0$. But it is also the case that $1^{\mathsf{inf}} = 1$ and $\texttt{compound}(\mathsf{inf}, 0) = 0$. Are there mathematical functions that distinguish between NaN and inf?

It turns out that there are! For example, the functions `minNum` and `maxNum` ([*IEEE 754-2008*, 5.3.1, p19]) take two arguments and return the min and max, respectively. They have the special, distinguished property that "if exactly one argument is NaN, they return the other. If both are NaN they return NaN."

With functions such as these, we can begin constructing the building blocks of more interesting functions (Figure 2). Unfortunately, $\texttt{maxNum}(a, b)$ and $a * b$ are not complete on their own; we additionally need at least a function $f(x)$ where $f(\mathsf{NaN}) = \mathsf{inf}$ and $f(\mathsf{inf}) = \mathsf{NaN}$. Does such a function exist? Yes! Several can be built from IEEE 754 primitives:

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/nand

$\texttt{maxNum}(a, b)$	NaN	inf
NaN	NaN	inf
inf	inf	inf

$a * b$	NaN	[illegible]
NaN	NaN	NaN
inf	NaN	inf

Figure 2: The behavior of some mathematical functions on our distinguished values NaN and inf. `maxNum` returns inf if either of its arguments is inf (some other functions have this property, like `hypot`). $a * b$ is inf only if both of its arguments are inf (there are many other examples, like $a + b$).

$$\begin{aligned}
f(x) &= \texttt{minNum}(-x, -1.0) + \mathsf{inf}\\
f(x) &= \texttt{hypot}(\mathsf{NaN}, \texttt{maxNum}(1/x, -\mathsf{inf}))\\
f(x) &= \mathsf{inf} - \texttt{maxNum}(x, 1.0)\\
f(x) &= \texttt{sqrt}(\texttt{copysign}(\mathsf{inf}, -x))
\end{aligned}$$

You can try these out in your favorite programming language, and if they don't work, your implementation is not IEEE 754 compliant. Why do these work? Let's take the first one, and compare NaN and inf for x:

$x = \mathsf{NaN}$	$x = \mathsf{inf}$
$\texttt{minNum}(-x, -1.0) + \mathsf{inf}$	$\texttt{minNum}(-x, -1.0) + \mathsf{inf}$
$\texttt{minNum}(-\mathsf{NaN}, -1.0) + \mathsf{inf}$	$\texttt{minNum}(-\mathsf{inf}, -1.0) + \mathsf{inf}$
$-1.0 + \mathsf{inf}$	$-\mathsf{inf} + \mathsf{inf}$
inf	NaN

Thinking of NaN as `false` and inf as `true`, we now have `AND` (`maxNum`), `OR` ($*$), and `NOT` ($\texttt{minNum}(-x, -1.0)$). With these we can create arbitrary functions $f(a_1, a_2, \ldots, a_n)$ that return our choice of NaN or inf for the 2^n different combinations of arguments. It is also possible to find more direct expressions that compute simple functions (Figure 3).

$\mathsf{inf} - \texttt{maxNum}(a + b, -\mathsf{inf})$	NaN	inf
NaN	inf	inf
inf	inf	NaN

$\texttt{abs}(\texttt{minNum}(b, -a) + \texttt{maxNum}(b, -\mathsf{inf}))$	NaN	inf
NaN	NaN	inf
inf	inf	NaN

$-\mathsf{inf}/\texttt{maxNum}(b, \texttt{maxNum}(a, -1))$	NaN	inf
NaN	inf	NaN
inf	NaN	NaN

Figure 3: Some interesting functions of two variables. They are isomorphic to the boolean functions `NAND`, `XOR` and `NOR` respectively, but more beautiful.

I found these functions through computer search,[4] using a

[1] This paper uses both exponents and footnotes extensively; please be careful of the difference.

[2] [*IEEE 754-2008*, 9.2.1, p44]

[3] [*IEEE 754-2008*, 9.2.1, p44]

[4] Source code is available at `https://sourceforge.net/p/tom7misc/svn/HEAD/tree/trunk/nand/`

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/nand/

parameter	**binary4**	binary16	binary32	binary64	binary128	binary
k, storage in bits	**4**	16	32	64	128	multiple o
p, precision in bits	**2**	11	24	53	113	k - round(4 * log 13
$emax$, maximum exponent e	**1**	15	127	1023	16383	2^{k-p-1} - 1
$bias$ = E - e	**1**	15	127	1023	16383	emax
$signbits$	**1**	1	1	1	1	1
w, exponent width	**2**	5	8	11	15	round(4 * $\log_2$(k)) - 13
t, trailing significand width	**1**	10	23	52	112	k - w - 1
k, storage width	**4**	16	32	64	128	1 + w + t

Figure 4: Parameters for the newly-introduced **binary4** encoding for IEEE 754, compared to the standard widths (see Table 3.5 in the standard[1]).

technique like bottom-up logic programming [5]. I start with a small set of constants, including arguments a and b, and then compute all of the expressions that can be made by applying a single mathematical function (e.g. abs(x), $-x$) or binary mathematical function ($x + y$, x/y, maxNum(x, y)) to existing expressions. The expression is actually a collection of values taken on for each possible substitution (in {NaN, inf}) to arguments a and b (i.e., it represents a function). If the expression has the correct values for each possible assignment to the arguments, then we are done. We only need to keep one (the smallest) expression that represents a distinct function, but note that we have to consider intermediate expressions that compute values other than NaN and inf. Also note that we need one of minNum, maxNum or copySign in order to compute the NOT function; we could think of these functions as therefore essential to mathematical completeness.

Particularly nice is inf − maxNum($a + b$, −inf), which returns inf if either of its arguments is NaN. We will call this the "NAN gate", for "Not NaN". The NAN gate is exciting because it can be used on its own to construct all other boolean functions! We can use NaN, inf, and this function to construct any computer and any computable function. Beautiful!

To program with numbers on computers, the real numbers are represented as strings of bits. Next we'll talk about efficient representations that allow us to manipulate NaN and inf with NAN gates.

2 The binary4 representation

IEEE 754 natively defines several bit widths for floating-point values, such as the 32-bit binary32 (aka "single-precision float") and 64-bit binary64 (aka "double-precision float"). The specification is parameterized to allow other bit widths; for example, half-precision 16-bit floats are common in GPU code for machine learning applications [7]. Smaller floats sacrifice precision, but require less space and allow faster calculations. For our purposes in this paper, since we only need to represent the two numbers NaN and inf, we are interested in the smallest possible representation.

This section describes the binary4 representation, a four-bit floating point number that is clearly allowed by the IEEE 754 standard.

The representation of any floating-point number has a single sign bit, some number w of exponent bits, and some number t of mantissa bits. For binary32, $w = 8$ and $t = 23$; and with the sign bit we have $23 + 8 + 1 = 32$ bits as expected. We need at least a sign bit, but what are the smallest permissible values of w and t?

$s\ E\ T$	value
0 00 0	+0
0 00 1	subnormal: $2^0 * 2^{1-2} * 1 = 1 * {}^1\!/_2 * 1 = 0.5$
0 01 0	normal: $2^0 * (1 + {}^1\!/_2 * 0) = 1$
0 01 1	$2^0 * (1 + {}^1\!/_2 * 1) = 1.5$
0 10 0	$2^1 * (1 + {}^1\!/_2 * 0) = 2$
0 10 1	$2^1 * (1 + {}^1\!/_2 * 1) = 3$
0 11 0	+inf
0 11 1	NaN
1 00 0	−0
1 00 1	−0.5
1 01 0	−1
1 01 1	−1.5
1 10 0	−2
1 10 1	−3
1 11 0	−inf
1 11 1	NaN

Figure 5: All 16 values representable in binary4 floating-point. The format works reasonably well even at this very low precision, although note how many of the values are not finite.

The most stringent constraint on w comes in [*IEEE 754-2008*, 3.4, p9], which states

> The range of the encoding's biased exponent E shall include:
>
> — every integer between 1 and $2^w - 2$, inclusive, to encode normal numbers
>
> — the reserved value 0 to encode ±0 and subnormal numbers
>
> — the reserved value $2^w - 1$ to encode ±∞ and NaNs.

E is the binary number encoded by w. It must include at least the two special values consisting of all zeroes and all ones (second and third clause). A conservative reading of "every integer between 1 and $2^w - 2$" seems to require that $1 \leq 2^w - 2$ (otherwise how could the interval be inclusive of its endpoints?), which would imply that w is at least 2. (However, see Section 2.1 for the hypothesized case where $w = 1$.)

The representation of NaN and inf are distinguished by the value of t when E is all ones. We certainly need to distinguish these, so $t = 1$ is the minimal size.

We have one sign bit, two exponent bits, and one mantissa bit, for a total of four. Since "single precision" is 32 bits, "half precision" is 16, 4 bits is "eighth precision." Given how nicely all this works out, shouldn't there be an `eighth` base type in most modern programming languages and GPUs? Since there are so few values representable, it would be practical for all the standard operations to be done in constant time via table lookups. All 16 possible values are given in Figure 4.

Four bits is not many, but is it possible to represent these two values more efficiently?

2.1 The hypothesized binary3 format

$s\,E\,T$	value
0 0 0	+0
0 0 1	subnormal: $2^1 * 2^{1-2} * 1 = 2 * {}^1\!/_2 * 1 = 1$
0 1 0	+inf
0 1 1	NaN
1 0 0	−0
1 0 1	−1
1 1 0	−inf
1 1 1	NaN

Figure 6: All 8 values of the hypothetical binary3 representation. There are no normal values; the only finite values are the positive and negative zero and a single subnormal which denotes 1 (or −1).

The IEEE 754 representation clearly requires a sign bit, and for this purpose we need at least one bit for the mantissa in order to distinguish NaN and inf. It is perhaps a stretch of the wording, but arguably the spec permits a 1-bit exponent ($w = 1$). To rationalize this we need to interpret the phrase "every integer between 1 and $2^w - 2$ inclusive" (that is, between 1 and 0 inclusive) as denoting the empty set. This seems reasonable.

With one bit for sign, exponent, and mantissa, we can represent just 8 different values. Here *emax* is 0, and the standard clearly requires $emin = 1 - emax$, so $emin = 1$. Certainly fishy for *emin* to be larger than *emax*, but we can just not stress out about it; the representable values are all reasonable-looking (Figure 6).

3 A hardware math accelerator

So now we know that we can build arbitrary computers with the `NAN` gate, representing the interconnects between the gates efficiently with binary3-coded real numbers. All that remains is an efficient implementation of the `NAN` gate itself. We could emulate such a thing in software, but software is much slower than hardware; we would also like to maximize the number of times that we can flip between states of the gate (the flip FLOPS) per second.

Fortunately, there are several pieces of hardware that implement IEEE 754 real numbers. I found a moderately-priced micrprocessor ($6.48/ea.), the STM32F303RDT6. This is a 32-bit ARM Cortex M4F processor with hardware floating-point running at 72MHz [6]. In the rather-difficult-to-solder 10mm surface-mount LQFP64 package, it has 64 pins, 51 of which can be used for general-purpose IO. Since a `NAN` gate using the binary3 representation needs 9 pins (3×2 for the inputs, 3 for the outputs), it is feasible to implement five `NAN` gates on the same chip with a few pins left over for jiggery pokery (Figure 7).

`https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/nand`

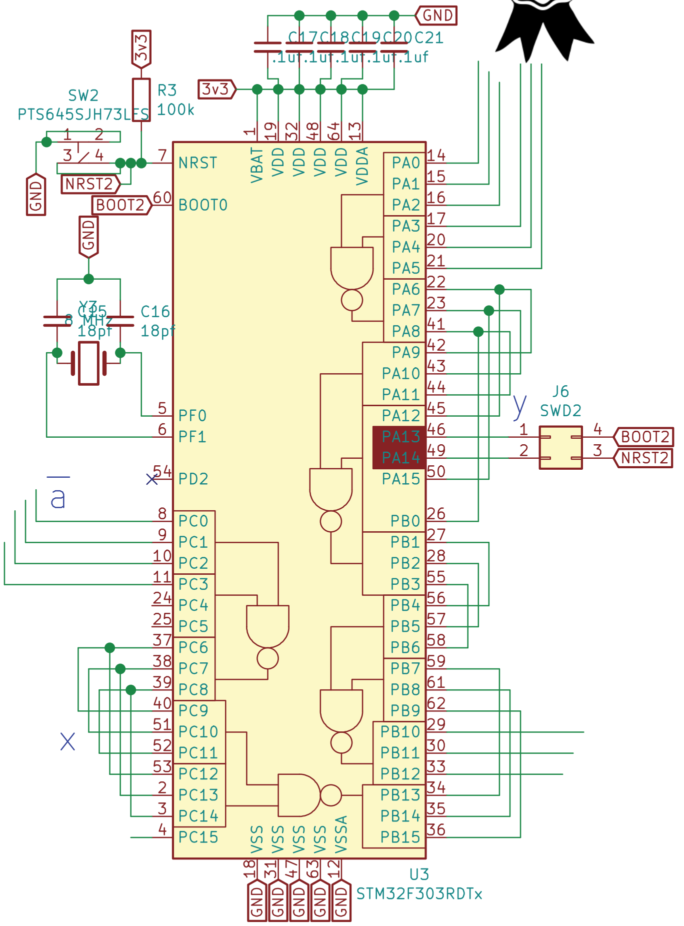


Figure 7: The STM32F303RDT6 wired up as 5 `NAN` gates, shown here *in situ*. This is a portion of a larger schematic. Also show is some support hardware needed for each microprocessor: A programming header, 5 filter capacitors, a crystal oscillator circuit, and a reset switch with external pull-up.

The hardware math accelerator itself can be thought of as a floating point unit (FPU), but one that is streamlined to run only a single instruction, the universal function $\mathsf{inf} - \mathtt{maxNum}(a+b, -\mathsf{inf})$. This is a function taking two binary3 real numbers and outputting a single binary3 number. Since there are only $2^6 = 64$ possible inputs, it can be straightforwardly implemented with table lookup, but this would require dozens of microprocessors, which might exceed our power budget. In fact there is significant structure to the function; for one thing, it can only return NaN or inf (even if arguments like -1.0 or 0.0 are given), and the binary3 representation of these only differ in one bit. Equivalent logic to determine that bit is as follows:

https://sf.net/p/tom7misc/svn/HEAD/tree/trunk/nand

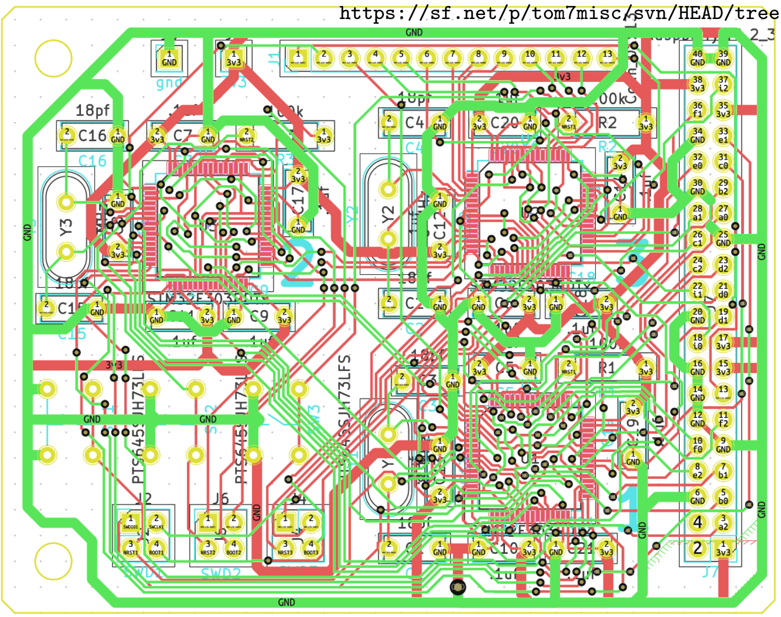


Figure 8: A beautiful hand-routed circuit board implementing a universal math accelerator, using only the universal NAN gate implemented with native floating point hardware.

```
if isfinite(a) && isfinite(b)
then
  // inf - (a + b) = inf
  0
else
  // a + b is nan when a is nan, b is nan, or a and b are
  // infinites with different signs. If they are both
  // -inf, then we have max(-inf, -inf) anyway, which is
  // the same as max(nan, -inf). So we have:
  if a == inf && b == inf
    // both positive infinities
    // inf - inf = nan
    1
  else
    // inf - -inf = inf
    0
```

So ultimately this function only returns 1 in the case that both inputs are exactly +inf, the pattern `0 1 0`.

If the inputs are a0 a1 a2, b0, b1, b2, and outputs are c0, c1, c2, then:

```
c0 = 0
c1 = 1
c2 = !a0 && !a2 && !b0 && !b2 && a1 && b1
```

So we can hardwire the outputs c0 and c1, and use the microprocessor-based NAN gates to compute c2 as a small boolean function.

Of course, each 0 or 1 above is actually itself a binary3-coded NaN or inf. Thus on the physical circuit board, this math accelerator has 2 × 3 × 3 input pins and 1 × 3 × 3 output pins. This is just shy of the total number of IO pins on the Raspberry Pi, so we use such a computer to drive the math accelerator. Given the large number of traces and small footprint of the microprocessors, routing the board gets somewhat involved (Figure 8).

As of the SIGBOVIK 2019 deadline, such a circuit board has been manufactured in China and is in possession of the author (actually the minimum order quantity of 10), but he is somewhat nervous about his ability to hand-solder these 0.1mm surface-mount leads, so we'll see how it goes! Please see `http://tom7.org/nan` for project updates or an embarrassing `404 Not Found` if I fail to reboot computing using the beautiful foundation of real numbers.

References

[1] 754–2008 IEEE standard for floating-point arithmetic. Technical Report 754–2008, IEEE Computer Society, August 2008.

[2] Floating-point extensions for C—part 4: Supplementary functions. Technical Report TS 18661-4:2015, ISO/IEC, 2015.

[3] JTC1-SC22-WG14. Rationale for international standard—programming languages—C. Technical Report Revision 5.10, ISO/IEC 9899, April 2003. http://www.open-std.org/jtc1/sc22/wg14/www/C99RationaleV5.10.pdf.

[4] Jim McCann and Tom Murphy, VII. The fluint8 software integer library. In *A Record of the Proceedings of SIGBOVIK 2018*, pages 125–128. ACH, April 2018. sigbovik.org/2018.

[5] Frank Pfenning. Bottom-up logic programming, November 2006. Course notes for 15–819K: Logic Programming.

[6] STMicroelectronics. STM32F303xD STM32F303xE. ARM®Cortex®-M4 32b MCU+FPU, up to 512kb flash, 80kb SRAM, FSMC, 4 ADCs, 2 DAC ch., 7 comp, 4 op-amp, 2.0–3.6 V, October 2016. Revision 5.

[7] Wikipedia. Half-precision floating-point format, 2019. https://en.wikipedia.org/wiki/Half-precision_floating-point_format.

We can confirm that the authors were willing to part with a very nice bridge at a considerable discount.

HonestNN: An Honest Neural Network "Accelerator"

Tim Linscott Vidushi Goyal Andrew McCrabb Pete Ehrett[*]
University of Michigan, Ann Arbor

ABSTRACT

It seems like everybody has been making their own neural network accelerator recently [1][2][3][4][5][6][7][8][9]. We figured we could do better. We kind of do. Probably. Anyway, neural networks are giant approximate systems with lots of repeated operations. We tried to have the hardware do some approximations of its own to make the computation go faster. And it does. Except that sometimes it will misclassify a picture of your cat as being a previously unknown painting by Van Gogh or something like that. I'd tell you about how much better we do, but our method for calculating performance is a little convoluted so that we look better. You should just skip to the Methodology section to read about it. But when we do use that method, we demonstrate a 15% "neural-adjusted performance" increase. Look, just go read the methodology, okay?

Keywords: Accelerators, Architecture, Machine Learning, ML, AI, DNN, CNN, Graph Processing, Blockchain, Cryptocurrency, 3D Printing, Security, Crowdsourcing, Computer, Electronics, Science, Engineering, #NovelResearch2019, #BuzzwordsGetCites, #BetterScience

1 INTRODUCTION

10^{-12}s after the big bang, a cooling universe allowed for the formation of the electron. Eventually, some of these electrons would find a new vocation when, around 550-600 million years ago, early bilaterians evolved a simple nerve cord. This opened the door to the process known as encephalization in which animals evolved brains and various sense organs. Most recently, an up-jumped primate at the forefront of the encephalization process figured out a way to compel those electrons to do neuron-like calculations, and branded it the "Artificial Neural Network." Recently, researchers have used this most recent turn of events for the 13.8 billion year-old electron to automatically generate memes and classify laundry [10, 11].

But in order to keep up with the universe's blindingly fast rate of progress, we all decided that we need to make those electrons *even more* efficient at generating memes and sorting laundry. This gave rise to the neural network accelerator. A bunch of researchers have made versions of these things.

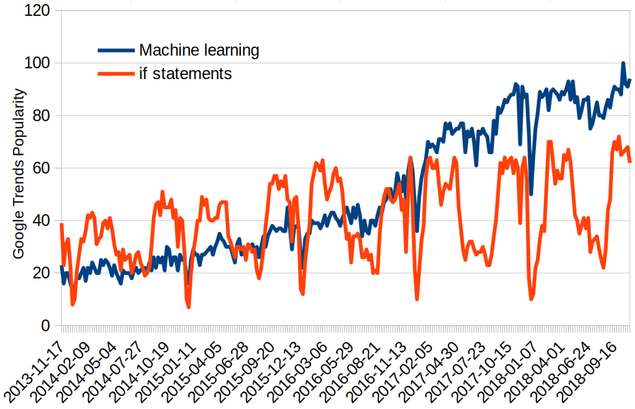


Figure 1: Ever since late 2016, Machine Learning has become more popular than if statements, according to Google Trends search data.

Though honestly, only a few have actually made the things. Mostly, we just run a synthesis job and do a simulation and call it a day. It's only because nobody else wants to put up with taping chips out that we can get this paper in.

We present HonestNN, a neural network accelerator where for once the grad students who actually worked on the project actually tell you what they actually did and didn't do. Our contributions are as follows:

- We tried to build and test a neural network accelerator.
- We develop a novel metric for evaluating accelerator performance that also happens to be the only metric in which we beat the state-of-the-art.
- We wrote a paper detailing our implementation and results. This is the big one.
- We cited a bunch of your papers, including one by the TPC chair.
- We express unparalleled honesty in the reporting of our results.

2 BACKGROUND

Machine Learning is an important paradigm in computer programming. According to Figure 1, since December 2016, there have been more Google searches for "Machine Learning" than for "if statements." Clearly, we should expect more ML in the future and fewer conditional branches. After all, when is the last time you read an "if statement accelerator" paper? Unless you count branch predictors, probably never. But what actually is machine learning? This section addresses that question.

[*] Pete is a CMU alum, which is how he knew we should send this to SIGBOVIK. He even has the ACH mug to prove it. Tim is currently affiliated with a company that would probably be embarrassed to know that he wrote this paper, so we'll just call it "Shmoogle".

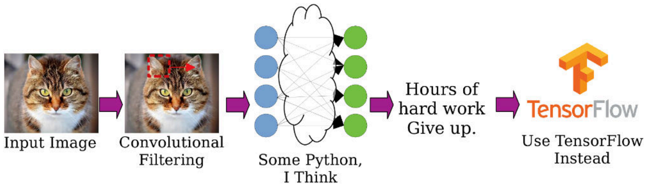


Figure 2: As far as I can tell, this is how state-of-the-art inference is done.

2.1 Modern ML Algorithms

There are a lot different ways to do machine learning nowadays. Mostly, though, we just use Deep Neural Networks (or "Machine Learning" in popular nomenclature) usually in the form of TensorFlow, though the research community tends to focus on how to use MLPerf as if that were a real thing. Most ML code tends to be of the form

```
import tensorflow

tensorflow.process("my_data.bin")
print("This is a picture of a cat.")
```

As the above code snippet shows, we can take advantage of the fact that almost every CNN or image recognition presentation that has a figure for image processing uses a picture of cat. Seriously, it's like a fundamental law of nature. To break down modern CNNs even further, Figure 2 shows how the process usually works. First, a picture of a cat is provided to the CNN. The CNN "convolves" the image using a processes called "convolution." Then there's some more code, which is probably written in Python, just like everything else is these days. You can either work really hard to create your own neural network, or you can just give up and use TensorFlow instead.

3 ARCHITECTURE

Maybe you guessed this from the last section, but I'm not that familiar with machine learning (seriously, when I review ML papers, I only put my expertise as a 2 if I'm feeling *really* confident.) But from what I hear, there are a lot of matrix and vector multiplications going on. And it's iterative or something, so you reuse the results from one step in the next step. So we have one cache to store the weights that we multiply with the features and another one to store the results. Then there are a ton of compute units. I was sure that I would need to do vectorized signed floating-point multiplication and some kind of threshold function, but I wasn't sure what else needed to be done in each step, so I thought I'd cover my tracks and just insert a complete RISC-V core in each of the compute units.

3.1 Optimizations

Our architecture takes advantage of real-time DNN pruning to problematically eliminate low-impact nodes. Real talk, though? We kind of forgot to connect a few of the compute units. But what are the odds that an important node is going to find itself in the broken pipeline? Pretty low, it turns out [12]. And the power we save not processing those probably unimportant nodes can get turned into increased clock speeds.

To improve transfer speeds between HonestNN and its peripherals, we use USB3.0 for maximum data rates.

I really wish this section was longer. I barely got HonestNN working in time to write this. There were bugs right up to the deadline, so I definitely did not have time to optimize anything. I think one of my source files has an `#ifdef USE_OPTIMIZATION` directive, and I'm 100% sure that `#define USE_OPTIMIZATION` was commented out by the end of my last debugging session.

3.2 Security Considerations

Everybody has been talking about security recently, so I assume that I need to address any potential reviewer comments for this architecture, too. The good news is that a lot of really smart people are coming up with a lot of novel security solutions. So for our threat model, I assume that we are to secure the part of the cloud that we want to perform our computation in. If the cloud is unhackable, then by deploying our chip in the cloud, we will be unhackable as well.

We plan on further ensuring the security of our solution using 1,048,576-bit, thousand-round RSA secured via the Blockchain, because we assume quantum computing will never really become a Thing. Also, because we haven't released our code yet (see Section 4.1), we also have something called "security through obscurity," which despite not being that good, is still something.

Real talk, though? I don't know what an attack on a neural-network accelerator would look like. It seems to me that attacks would all be higher up in the stack since HonestNN only exists to do math really fast. It would be like attacking a calculator. That said, malware targeting calculators is a real thing, so maybe I should be worried. [13]

4 METHODOLOGY

4.1 Framework

We wanted to implement HonestNN in SystemVerilog, but were told by people paid more than us (who may or may not be last authors of this paper), to implement it in VHDL. A week later, these same so-called "authors" changed the requirement to C++, then Python, then probably HTML (I fell asleep during that meeting), and then finally SystemVerilog.

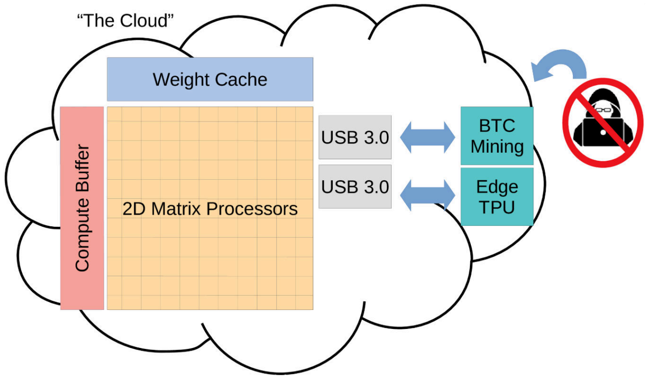


Figure 3: A diagram of how our accelerator works. Our architecture lives in an "unhackable" "cloud." The Edge TPU is included in the event that the rest of the accelerator isn't working, but if Google won't give us one of those, we will fall back on mining Bitcoin to pay for this project.

We're pleased to report that we started in SystemVerilog from the beginning, so there was no lost time.

In the end, we implemented HonestNN in SystemVerilog. By "we" I obviously just mean the grad students on this paper, *i.e.,* the entire remaining author list since one of us got annoyed and took off everyone who didn't actually do the work. When's the last time you've ever heard of a professor writing code for a research project? And by "implemented" I mean prototyped. By "SystemVerilog," I mean to say that I wrote the various components so that I could synthesize everything in the end. Okay, not "everything," per se; we used CACTI [14] for the caches and stuff, because I couldn't figure out how to get those to synthesize from the Verilog. For testing, I wrote this great simulator that shows how everything is supposed to work in a mostly cycle-accurate way. I did that in Python, so it took about a day to write and five days to run. I think I used a little bit of BookSim as well [15]. Or at least, I opened it once. And how can I forget gem5? Any architecture paper that doesn't mention a gem5 simulation is crap. *You* may have used real hardware but nothing beats "cycle-accurate simulation." And if you believe I actually even went as far as doing any of the above, I have a bridge to sell you. Why do you think my "code" isn't on GitHub? Hey, I'm citing your papers; don't judge me.

So how do we turn this pile of overlapping implementations into meaningful, comparable results? If it weren't for the fact that I promised unrivaled honesty in this paper, I'd probably just mumble something about "benchmarking" before suggesting we take the conversation offline. The short version is that we use that Python simulator I mentioned. But I'll have you know that I wrote that simulator as a proof-of-concept within the first month of the project and only touched it again before the deadline when I realized that nothing else was going to give me meaningful performance numbers. The simulator assumes an idealized neural network perfectly fitted to our design and fully utilizing all its compute without thrashing the cache. Then it outputs a single number representing the number of operations performed by each compute unit, which gets fed into another script that I wrote last night to turn that into neural-adjusted performance, which we discuss below.

4.2 Neural-Adjusted Performance

We define "Neural-Adjusted Performance" as GOPs per Watt per dollars of NRE normalized against process node, algorithm, and years since the release of TensorFlow v1 (lower is better). Let's walk through each of these in turn. First, GOPs are counted by the number of irreducible mathematical operations performed by the accelerator in the best case where the data exactly fits the pipeline width and all nodes are in use and no lanes have accidentally been disconnected from the rest of the architecture. Second, we normalize with respect to power, because that's the limiting factor in new architectures according to my advisor [16]. Third, we normalize against the Non-Recurring Engineering expenses that went into developing this chip. Since the chip was designed entirely by underpaid grad and undergrad students desperate to get the project done before the last of the grant money dried up, it cost less than $100,000—much less than the average industry-made chip. We normalize against process node to balance out the fact the we spent 10 months trying to file paperwork that would get us access to real-world standard cell libraries and getting ignored by university and industry bureaucrats so many times that we finally gave up and used a really terrible open-source standard technology library. Not that I'm bitter or anything. We also normalize against the algorithm used. While we working on HonestNN, some people came out with a bunch of cool techniques in software that would make neural networks way faster. We figure that if we'd known about them, our core would be a lot faster, so we factor in the speed and accuracy of the fastest state-of-the-art neural network at the time the paper came out. But we also need to give credit where credit is due. Later researchers are really just poaching good ML ideas from TensorFlow, so the earlier researchers had a leg up on us in that they were forced to be creative and smart and come up with novel ML techniques on their own. If we weren't afflicted by analysis paralysis due to being surrounded by so much ML literature and so many ML APIs and ML papers, we'd probably have come up with some even better ideas of our own. So, by normalizing against the number of years since TensorFlow v1 was released, we attempt to level the playing field between us and the researchers of yesteryear.

All in all, this yields a broad perspective on the "is it worth it to spend this amount of time and money on improving automated laundry categorization by 0.1%?” question that all developers and researchers ask themselves at some point.

4.3 Benchmarks

We tested our neural network accelerator using SPEC2017 as well as a basic Hello World program. We also tried to use MiBench, but it was way too hard to port without knowing the rationale behind the hard coded values (yes, you heard it right), so we gave up there.

Next, we used LinPack. Have you heard of this one? It’s like a ten million by ten million matrix-matrix multiply. No one in the real world would ever need to do a ten million by ten million matrix-matrix multiply. But this is academia and not even my advisor knows what’s going on inside real-world machine learning, so I guess this means that this is our best guess. But, hey, LinPack is great. It rewards ~~stupid~~ elegant architectures like HonestNN that are pretty much just bundles of compute cores. No one should be the least bit surprised that we did pretty well here.

Cloud-based neural network workloads demand that providers balance handling large volumes of requests with the need to serve responses at high speeds. Thus, neural network accelerators need to be designed around a conscious bandwidth-latency trade-off. We designed our accelerator to be easily customizable to optimize for one design point or the other. We analyzed a comprehensive selection of benchmarks targeting bandwidth and latency goals with a wide range of possible configurations of the accelerator. Figure 4 presents a pareto curve of the optimal accelerator configurations.

Look. My advisor wrote that paragraph just a few minutes before the deadline. I don’t want to take it out, but in the interests of honesty, I’m going to repeat what was just said with a few corrections. Here goes:

Cloud-based neural network workloads demand that providers balance handling large volumes of requests with the need to serve responses at high speeds (*pretty true*). Thus, neural network accelerators need to be designed around a conscious bandwidth-latency trade-off (*but this is definitely not a knob you can easily tune*). We designed our accelerator (*my advisor still doesn’t remember the project’s name*) to be "easily" customizable to optimize for one design point or the other (*I literally had an undergrad in tears trying to configure this thing*). We analyzed a comprehensive selection of benchmarks (*Hello World and LinPack*) targeting bandwidth and latency goals with a wide range of possible configurations of the accelerator (*five configurations in total*). Figure 4 presents a pareto curve of the "optimal" accelerator configurations (*yeah, right*). One of my co-authors (I’m not sure who, because we’re all editing this at once) keeps editing this section to complain how that’s not really a pareto curve, but it’s 2 hours before deadline and I literally don’t care anymore.

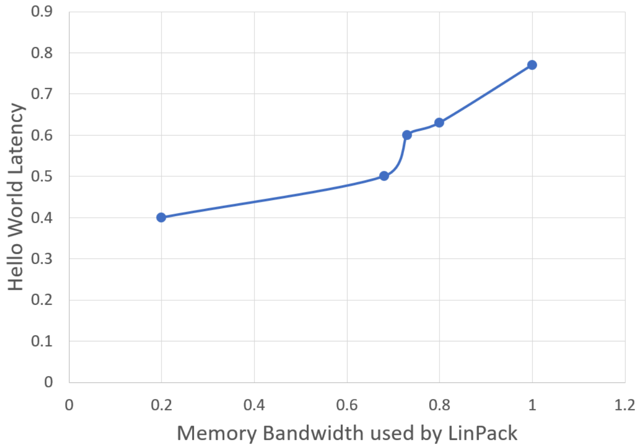


Figure 4: HonestNN can be configured to prefer latency (time to execute Hello World) or bandwidth (memory bandwidth consumed running LinPack). I’m not really sure what the take-away is here other than "Hey, look! A pareto curve! These guys clearly did a thorough analysis!" (But wait, no, that isn’t actually pareto, is it...)

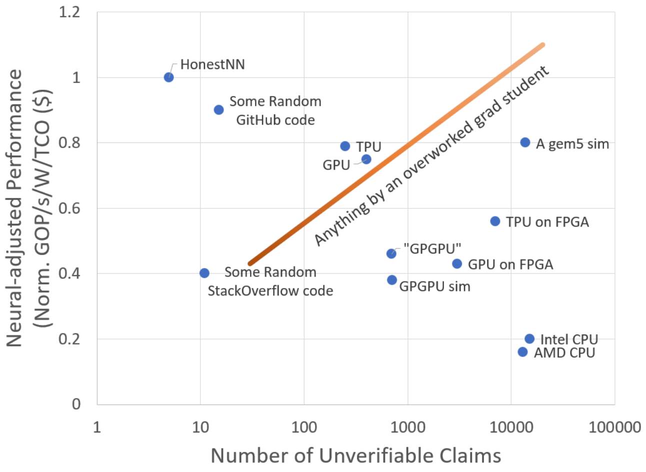


Figure 5: HonestNN beats the state-of-the-art on both neural-adjusted performance and the honesty with which we present our results.

5 RESULTS

5.1 Performance

Figure 5 compares HonestNN to the state-of-the-art research and industry solutions and to random internet code. But did you know that it’s really hard to compare your research project to other people’s research projects? It’s not like they share the data behind their graphs. So all those research papers *that I was going to cite if I’d had access to some useful data for once* all contained large numbers of claims that I can’t verify. Oddly, I found that a research paper’s performance was directly correlated to the number of dubious claims made by the authors. As for industry solutions, it's not like NVIDIA is going to give the source code to their latest GPU. Still they have some insanely dense manuals that are less fun to read than your average angry adviser email and which make me wonder whether any of the engineers even understood the design docs at all. On the other hand, we also looked into a number of sources of random internet code some of which has been elevated to the level of divine revelation by academics. gem5, for instance, which brings nightmares and madness to any foolish enough to study its unfathomable eldritch mysteries, is considered the gold standard in computer architecture simulation. However, it is built from 13,716 different commits at last count, none of which make any sense. Meanwhile, you can find some random code on StackOverflow that accelerates neural networks as long as you're willing to assume that "the memory subsystem can't be that complicated" and won't play into performance too much.

Dear reader, we make no such assumptions, which is why HonestNN leads the pack with fewer unverifiable claims than the state-of-the-art and the internet at large. The reason for this is simple: we do not claim that our accelerator actually works at all, nor do we claim that if it did work, you would actually want to use it. In fact, I'll go out on a limb and claim that neither of these is the case at all. But one thing's for certain: its neural-adjusted performance normalizes to one.

5.2 Design Costs

To determine the area and power of our design and its maximum clock speed, we synthesized our implementation using Synopsys Design Compiler. Turns out that Design Compiler is really, really, *really* slow and that it gives different numbers every time you rerun synthesis. We basically just launched a bunch of different synthesis runs with random edits made to our synthesis script and picked whichever one had completed and gave us the best results by the time this paper deadline rolled around. Honestly, I'm not sure if Design Compiler just stumbled upon a pure genius optimization that time or whether this is an erroneous result. Because there was this one other synthesis job that gave me the same clock frequency but at like 5× more area. Next time we'll just wave the rubber chicken of `compile_ultra` and call it a day.

In the end, though, synthesis results really aren't *that* important since we're going to normalize everything in our performance analysis just like the other cool researchers do [17]. In short, our accelerator is of size 1, consumes 1 unit of power, and operates at a clock speed of 1.

5.3 Development Cost

We hired a bunch of underpaid world class labourers (read graduate students), some apprentices (read undergraduate interns), and finally used the latest "up-to-date" free tools provided by the university to build our awesome accelerator–HonestNN–which beats the so-called "state-of-the-art" NN accelerators from industry built by overpaid engineers and researchers with the most sophisticated technology. Such a waste of resources! Not to mention, by the end of this project, the lead author of this paper was roped in by the leading software giant because of his *sheer* expertise in the area. Well, knowing that the fate of the world is in *honest* hands, I can finally rest in peace.

6 LIMITATIONS

In the interests of full disclosure, we present the weaknesses associated with HonestNN.

- There's really nothing concrete I can point to and say "this is HonestNN." I mean, there are a bunch of scripts and log files, but there's really no single source of truth here.
- I never really verified the functionality of HonestNN. I ran a few tests on a couple Verilog modules and the simulator works "fine" (see below), but I don't think that really counts.
- I've found bugs in the simulator and reran the results more times than I can count. Sometimes fixing the bugs made HonestNN faster, sometimes it made it slower. Did I catch all the bugs? I really don't know anymore. But the deadline is looming large, so I guess I have to act like I did.
- I was really nervous about the whole "untested chip" thing, so I snuck a TPU into our design. I don't know what the rest of HonestNN is needed for if we have a TPU on hand. Why not just use the TPU?
- If anyone looked at my code, I would either be blacklisted from every major tech company from now until the singularity occurs or forced to memorize every known style guide before I was even allowed to *look* at a computer again.

7 RELATED WORK

Okay, so there are a bunch of academics who thought of ideas for neural network accelerators. But there's no way I'm going to be able to cite them all. Seriously, NIPS had like 3,000 submissions this year. So I'm just going to cite the top nine papers from Google Scholar [1–9], and another paper from my advisor's best friend's former postdoc who gave a really memorable talk or something [18]. I'm not totally sure how the implementation on this one worked, but it seemed interesting and I'm including it anyway [19]. But none of these people have a hundred billion dollars. You know who does have a hundred billion dollars? Google. And what do you know? They made their own neural network accelerator, too. And actually built it. And we can actually use it. But they're not academics, so all I'm giving them is this one lame citation: [20].

If you're one of the blind reviewers and you're trying to find the authors of this paper in related work or metadata, let me spare you the effort and include some egregious and unwarranted citations of our own papers so that it's easy to figure out who we are: [21, 22].

8 FUTURE WORK

At some point, we should probably build a working version of HonestNN that really does run from start to finish instead just being a delicately balanced pile of scripts with all the structural integrity of a house of cards. My advisor wants to have the next version use emerging NVM technologies, and our sponsors want to know whether our work has any applications to autonomous driving.

Personally, if this paper gets in, I'm never coming back to this project. I need novel ideas to fill out my thesis, and incremental improvements on this dead horse of a project won't make the cut. Besides, no one really gets around to their future work anyway. Do you?

9 CONCLUSIONS

Who cares? By the time you read this, something else will have come along that supersedes everything we built for this paper. Or that did the analysis better. Or that faked their results to make it look better. Or the singularity will have occurred. Whatever.

But before you think of rejecting this paper out of hand because of its shoddy implementation and weak evaluation methodology, why don't you take a minute to talk to your grad students and make sure this isn't happening under your nose right now? Go ahead, I'll wait. But you can be sure of one thing: HonestNN is the most honest neural network accelerator you'll ever see published. Please?

REFERENCES

[1] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, 44(3):14–26, 2016.

[2] Ryuichi Sakamoto, Ryo Takata, Jun Ishii, Masaaki Kondo, Hiroshi Nakamura, Tetsui Ohkubo, Takuya Kojima, and Hideharu Amano. Scalable deep neural network accelerator cores with cubic integration using through chip interface. In *SoC Design Conference (ISOCC), 2017 International*, pages 155–156. IEEE, 2017.

[3] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. Yodann: An ultra-low power convolutional neural network accelerator based on binary weights. In *ISVLSI*, pages 236–241, 2016.

[4] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. A high-throughput neural network accelerator. *IEEE Micro*, 35(3):24–32, 2015.

[5] Laura Fick, David Blaauw, Dennis Sylvester, Skylar Skrzyniarz, M Parikh, and David Fick. Analog in-memory subthreshold deep neural network accelerator. In *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4. IEEE, 2017.

[6] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. Accelerating deep convolutional neural networks using specialized hardware. *Microsoft Research Whitepaper*, 2(11), 2015.

[7] Ryuichi Sakamoto, Ryo Takata, Jun Ishii, Masaaki Kondo, Hiroshi Nakamura, Tetsui Ohkubo, Takuya Kojima, and Hideharu Amano. The design and implementation of scalable deep neural network accelerator cores. In *Embedded Multicore/Many-core Systems-on-Chip (MCSoC), 2017 IEEE 11th International Symposium on*, pages 13–20. IEEE, 2017.

[8] William J Dally, Angshuman Parashar, Joel Springer Emer, Stephen William Keckler, and Larry Robert Dennison. Sparse convolutional neural network accelerator, February 15 2018. US Patent App. 15/458,837.

[9] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *ISCA '16*. IEEE, 2016.

[10] V Peirson, L Abel, and E Meltem Tolunay. Dank learning: Generating memes using deep neural networks. *arXiv preprint arXiv:1806.04510*, 2018.

[11] Li Sun, Simon Rogers, Gerardo Aragon-Camarasa, and J Paul Siebert. Recognising the clothing categories from free-configuration using gaussian-process-based interactive perception. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 2464–2470. IEEE, 2016.

[12] Who_really_cares. xkcd: Machine learning. In *Does-anyone-read-this-part-anyway?*, 2017.

[13] Piotr Bania. Gaara, 2007.

[14] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. Cacti 6.0: A tool to model large caches. In *International Symposium on Microarchitecture*, 2007.

[15] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim. A detailed and flexible cycle-accurate network-on-chip simulator. In *ISPASS '13*, 2013.

[16] Todd Austin. Application-specific design tutorial. In *University of Michigan EECS 573: Microarchitecture*, 2018.

[17] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 367–379. IEEE Press, 2016.

[18] Alessandro Aimar, Hesham Mostafa, Enrico Calabrese, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Iulia-Alexandra Lungu, Moritz B Milde, Federico Corradi, Alejandro Linares-Barranco, Shih-Chii Liu, et al. Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps. *IEEE transactions on neural networks and learning systems*, pages 1–13, 2018.

[19] Doug Zongker. Chicken chicken chicken: Chicken chicken. *Annals of Improbable Research*, pages 16–21, 2006.

[20] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA '17*, pages 1–12. IEEE, 2017.

[21] Timothy Linscott, Pete Ehrett, Valeria Bertacco, and Todd Austin. Swan: mitigating hardware trojans with design ambiguity. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7. IEEE, 2018.

[22] Pete Ehrett, Vidushi Goyal, Opeoluwa Matthews, Reetuparna Das, Todd Austin, and Valeria Bertacco. Analysis of microbump overheads for 2.5 d disintegrated design. 2017.

Simultaneous Microwaving Architectures: An Efficient Scheme for Multiplate Heating

Bottom Text

Charles Yuan[*]
Technology Evangelist
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA
charlesyuan@cmu.edu

Abstract

Traditional food processing technology often encounters a performance bottleneck with regards to throughput of heating. Stoves, ovens, and handheld flamethrowers have all been applied to the task of efficient food heating, with varying results. Here we present Simultaneous Microwaving (SMW), an alternative architecture for the warming of plate-based foods, and also generalize it to other categories of consumables. We evaluate the limits of the parallelism introduced by SMW and compare it to other state-of-the-art techniques. We conclude that SMW is a suitable design for the implementation of highly energy- and time-efficient food warming systems.

Keywords High-Energy Physics, Family and Consumer Sciences, Computer Architecture, Dependent Types, Machine Learning, Byzantine Fault Tolerance

ACH Reference Format:
Charles Yuan. 2019. Simultaneous Microwaving Architectures: An Efficient Scheme for Multiplate Heating. In *Proceedings of ACH Special Interest Group on Harry Q. Bovik (SIGBOVIK'19)*. ACH, Pittsburgh, PA, USA, 5 pages.

1 Introduction

From the beginning of time, people have always been interested in cooking their food. Since the dawn of agriculture, many techniques of food preparation have been devised. In the past few millennia, cooking has advanced to roasting over a campfire to self-timing induction cooktops, even as the underlying principle of food preparation has remained the same.

Cooking has several advantages that make it an attractive process. Food that has been cooked is more nutritious, more likely to be safe from pathogens, is easier to digest, and has an improved flavor profile. However, cooking is a time-consuming process that often requires several hours of continuous attention from human operators on a daily basis. Automating the preparation of food therefore has the potential to save billions of person-hours of time per year.

The microwave oven (MWO, for short) was developed in 1947 and is widely regarded one of the most influential food preparation technologies of contemporary engineering. Derived from early investigations into electromagnetic radiation in the twentieth century, including the development of radar, the potential of using radiation to heat food was quickly exploited and marketed to the masses. Modern MWOs are now found in even the most basic of households, and are regularly operated by unskilled individuals such as children. It is difficult to imagine the modern culinary environment without the microwave oven.

1.1 MWOs

The operating principles of the MWO are conceptually simple: a cavity magnetron powered by an electric power supply emits 2.45 GHz microwave radiation into the interior of the apparatus. Reflective surfaces inside the inner compartment deflect the radiation until it strikes the target food medium. Polarized molecules within the food are excited by the radiation and gain kinetic energy, eventually increasing the overall temperature of the food medium. An image of a typical microwave oven follows in Figure 1.

For those readers who may be unfamiliar, the simplified operation of the MWO is as follows:

1. The door of the MWO is opened, usually by depressing a conspicuous button at the lower-right corner of the front face of the apparatus.
2. Any existing contents of the MWO are removed, except for a circular plate at the bottom; this is commonly known as the “turntable”.
3. Food to be heated is placed on the circular disk.
4. The door is restored to its original closed position.
5. The electronic interface of the apparatus is used to select parameters of heating. The parameters vary highly

[*] Made you look.

SIGBOVIK'19, April 04, 2019, Pittsburgh, PA, USA

Figure 1. A typical microwave oven.

depending on the manufacturer and model of the device; common parameters include duration of heating and a power level.

6. A user interface element usually labeled "start" is activated and the heating begins, continuing until an alarm (usually auditory) is triggered.
7. At this point the heating is complete. The door may be opened and the prepared food removed. The door should then be once again closed.

As is evident, correct usage of the MWO is burdensome and carries much overhead especially for inexperienced operators. Large-scale food processing requires minimization of the total time and energy required to complete the process. In this paper we will not focus on energy as it is generally accepted that a higher power rating for the MWO decreases heating time though in a nonlinear fashion. We will refer the interested reader to further reading on this subject later in the report. For now, we assume a fixed power level for the MWOs we consider.

1.2 Processing Time

The total processing time t_{MW} for a volume of food may be computed as:

$$t_{\text{MW}} = t_{\text{fixed}} + t_{\text{variable}}$$

where t_{fixed} is the fixed time overhead and t_{variable} is the variable component of processing. Fixed contributors include the time to open and close doors and (with a small degree of error) the time to specify parameters of heating. They will be treated as constant for the purposes of our analysis.

The variable component may be further broken down:

$$t_{\text{variable}} = t_{\text{load}} + t_{\text{heat}}$$

where t_{load} is the time to load and unload the contents of the MWO, and t_{heat} the time duration when the magnetron is activated and the food is being heated. The second generally dwarfs the first, and usually especially dominates the fixed component.

2 Sequential Operation

Consider the batch processing of n items of food $x_1, x_2, \ldots, x_n$. The best-known sequential algorithm to heat all units of food follows in Algorithm 1.

```
Result: Output vector $\text{out}_i$ is the heated result of $x_i$
for $i \leftarrow 1$ to $n$ do
    openDoor();
    insert($x_i$);
    closeDoor();
    setParams();
    pressStart();
    wait();
    openDoor();
    $\text{out}_i \leftarrow$ remove();
end
```

Algorithm 1: Naive sequential algorithm

The time complexity of the algorithm is

$$n \cdot t_{\text{fixed}} + \sum_{i=1}^{n} t_{\text{variable}}(x_i)$$

However, in practice this process is inefficient for two reasons. The first is that the fixed cost is repeated for every food item when it may be possible to amortize it over batches of inputs. The second is that the maximum capacity of the MWO is normally not reached in the sequential approach. If we process several food items at once, we would alleviate both of these concerns. However, how this form of parallelism may be achieved in general is not obvious.

3 Difficulty of Parallelization

To see how parallelism may be challenging in practice, we consider the structure of a typical food item in a batch workload. An item consists of *food medium* held by a *food container*. The container can take many forms, ranging from polystyrene foam or paper boxes to ceramic bowls. One particularly common container is the *plate*, a flat, circular disk structure usually made from paper, plastic, or ceramic material bent gently upwards at the edges. The plate has pleasant topological properties that make it a popular choice for many types of food.

Suppose we wished to insert and heat multiple items in every iteration of the outer loop of Algorithm 1. Since plates may not physically overlap, only a limited number of plates may be placed on the turntable of the MWO. To minimize the number of total iterations of the loop, plates must be packed in an optimal configuration each iteration. This is an instance of the well-studied packing problem. Given a set of plates of varying sizes, determining the maximal number of

Figure 2. Arrangement of plates in space.

plates that will fit on the turntable is **NP**-hard. Furthermore, the greedy approach of using locally maximal fits into each iteration does not guarantee a globally optimal arrangement of plates, as the sequence of plates may need to be rearranged for an optimal batch process. Two levels of optimization are thus required in some sense, and determining the optimal arrangements is computationally complex and infeasible. A diagram of the arrangment of plates on the turntable is contained in Figure 2.

There is a further problem: even with an optimal arrangment of plates on the turntable, the vast majority of space within the MWO is wasted since the space above the plates is left empty. In practice, heuristic and approximate algorithms are used to arrange plates on the turntable each iteration, achieving respectable but less-than-optimal results. However, almost no implementations of MWO operation utilize available capacity along the vertical dimension. This optimization is the basis of our new proposed architecture.

4 Simultaneous Microwaving

We now introduce the Simultaneous Microwaving (SMW) architecture as an evolution of traditional MWO management algorithms. We first make several assumptions that hold true on almost all practical MWO batch workloads:

1. All food items are held by plates whose size does not exceed that of the turntable.
2. The height of each food item above the plate is bounded by a constant c which is less than a ratio $1/k$ of the height of the heating compartment, where $k \geq 2$. We denote k the *plating count*.
3. Plates are made of a material resistant to vertical compression, such as hard plastic or ceramic.
4. The thickness of a plate is negligibly small compared to c.

The key innovation of SMW is the utilization of available vertical space. Vertical arrangement of plates is normally impossible due to undesirable contact of foodstuffs with contaminants from above. However, it is possible to safely stack plates vertically through the use of *interlock plates*: upturned plates inserted between each vertical stage. The interlock plates protect food from contact with any surface other than the concave side of a plate. The architecture is depicted in Figure 3. We refer to the physical stack of plates as the *pipeline*, each pair of food item and interlock plate as a *task*, and the processing of each stack as a *cycle*.

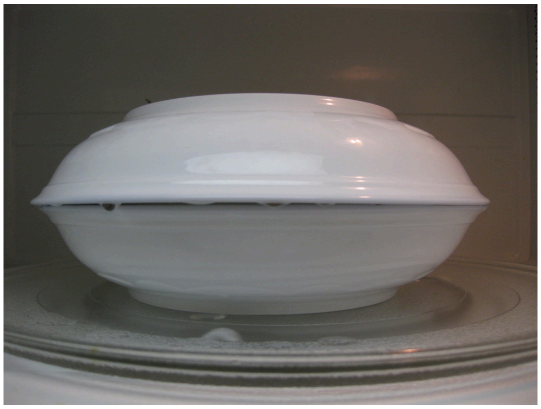


Figure 3. Physical stacking of plates within the heating pipeline.

With the assumptions we have made in place, the number of food items that may be processed per iteration is at least k. The parallel time complexity of the food processing algorithm is thus

$$\frac{n}{k}\left(t_{\text{fixed}} + \max_{1 \leq i \leq n} t_{\text{variable}}\right)$$

which is a significant improvement over the sequential algorithm.

5 Task Scheduling and Pipelines

The simplifying assumption that the height of each plate is bounded by a constant means that the parallel speedup is attainable regardless of plate arrangement. Of course, this restriction may be relaxed and scheduling techniques applied to increase the utilization of space. Though we will not focus on that computationally complex problem in this paper, we acknowledge it as a possibility.

For now, we address a critical difficulty in the management of SMW schemes of heating, namely *pipeline stalls* when some food items take longer to heat than others. Suppose of the plates in one cycle, one plate takes significantly longer to process than the others. And in the worst case, this plate resides at the top of the stack. Since the stack is a LIFO structure, it is impossible to safely remove tasks that have finished heating before this task at the top, without iteratively popping from the stack in linear time. In the worst case, tasks may happen to be arranged in reverse order of heating time required, causing this reshuffling to take quadratic time. Therefore, we simplify the architecture by only inserting and removing entire batches rather than manipulating individual tasks in the stack with finer granularity. This has the effect of possibly overheating certain food items, which for our workflows is generally tolerable but may be a concern for some operators.

Since we take a coarse-grained approach each cycle, the duration of a cycle must be equal to the maximum of the required durations of each item. If it were possible to predict heating time required, this issue could be avoided, but such prediction is impossible in general. It is possible within our implementation to annotate each task with a hint, inserted automatically by the food manufacturer, in order to improve the balancing of fast and slow tasks. This information is usually printed on the documentation of the foodstuff. Such duration hints are valuable but are not universally accurate or trustworthy, so we must take the conservative approach of assuming we have zero information available before heating begins.

6 Limits to Parallelism

Ideally, a SMW system would achieve speedup of at least a factor of k over the sequential algorithm. However, in practice this does not occur for a very important reason. Since microwaves do not propagate after they impact a food medium and impart kinetic energy, increasing vertical space utilization in the MWO reduces the relative heating efficiency of the entire system and causes an effect of diminishing returns. The remainder of this paper will thus focus not only on the efficiency of implementing the parallel stacking of plates, but also the effect of decreased heating energy available to each task. A trade-off must carefully be considered, and for some systems, it is counterintuitively the case that a somewhat reduced vertical space utilization leads to overall faster heating time.

7 Implementation

We did not implement the system due to lack of time before the conference submission deadline. If we had, we would have used a 1000 W microwave oven and a wide variety of common foods amenable to microwave heating. We would have conducted the following experiments and concluded the results following thereafter. Naturally, since we did not perform any of the experiments, the following data are all fabricated.

First, we evaluated the overall heating time of several food workloads under sequential heating and also under a SMW architecture. We evaluated using the following food benchmarks. Each benchmark was calibrated to have a plating count $k = 4$.

- Bulk white rice, pre-cooked;
- "Cup" ramen noodles, chicken flavor, filled to line;
- Spaghetti with Alfredo sauce;
- Italian-style beef meatballs with breadcrumb coating;
- Dry Froot Loops® brand breakfast cereal;
- Water buffalo entrails;
- Crushed glass and pig blood;
- Literally just mayonnaise.

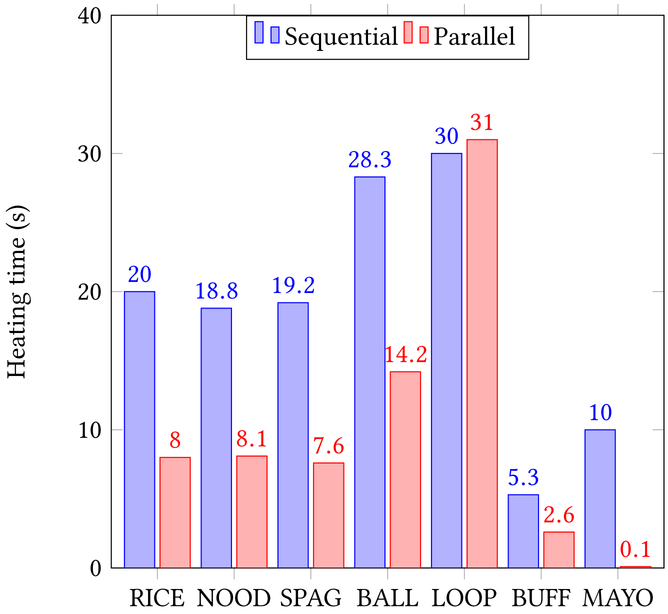


Figure 4. Comparison of sequential and parallel heating for each food type.

Using the first three benchmarks, we also adjusted the degree of stacking used, from 1 (sequential) to 4 (maximally parallel). We evaluated the relative speedup of the different options.

Finally, we evaluated the impact of plate material on heating efficacy under the maximally parallel scheme. We used polycarbonate, porcelain, clay, and silver-coated brass platings.

8 Evaluation

Figure 4 shows the heating comparison. We were able to achieve at best a 2.5× speedup over sequential heating, for the rice and spaghetti tasks. Other food types, such as meatballs, and buffalo entrails, saw somewhat more modest speedup of approximately 2.0×. Unfortunately, we were disappointed with our results for the dry breakfast cereal, which actually slowed down under SMW heating. The crushed glass task failed to heat appreciably under either regime and was excluded from the final results. Finally, we are amazed by the performance of the mayonnaise, which promptly vaporized after less than one second under SMW heating.

Figure 5 depicts the relative speedup as the stacking degree increased for the bulk starch tasks. It appears that increasing stacking degree to 2 nearly doubled the speed of heating. However, increasing it to 3 had a more modest improvement, and increasing it to 4 had almost no effect at all. It is clear

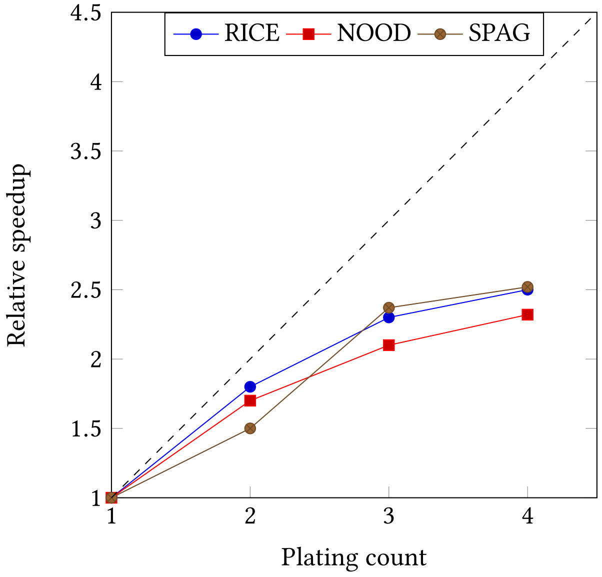


Figure 5. Relative speedup as degree of stacking increases.

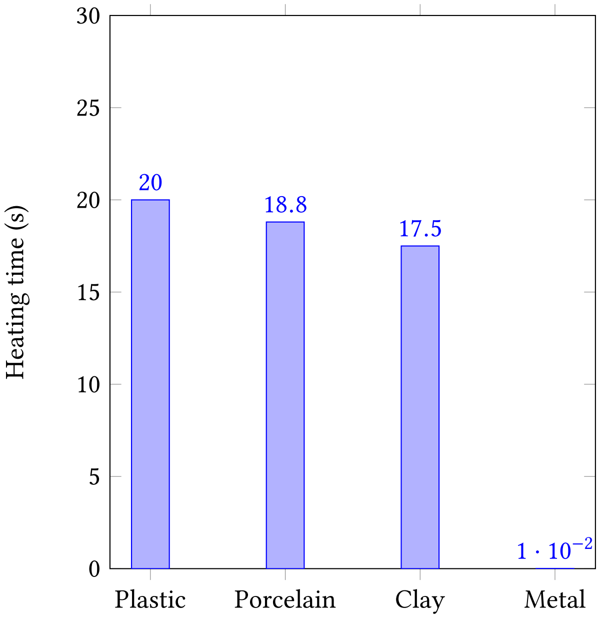


Figure 6. Effect of plate material.

that the bottleneck of microwave penetration bandwidth is reached relatively quickly.

The last study, Figure 6, shows the effect of plate material. The choice of plastic or ceramic plate material appears to have had no significant effect on heating efficiency. Upon the fourth trial with silver-coated brass plating, a large blue arc appeared inside the MWO and caused a large electrical fire which took approximately one hour to contain and destroyed the experimental appratus. We assume the food was cooked in that circumstance, though we were unable to recover the sample.

9 Conclusion

We have introduced Simultaneous Microwaving (SMW), an effective mechanism for improving the work efficiency of heating common foods using microwave ovens. It is relatively simple to implement and has been demonstrated to present marked improvements for a wide variety of food workloads. Despite the presence of a significant performance bottleneck of microwave penetration, systems that use a modest degree of SMW parallelism observe a large increase in heating efficiency. We believe this technology is an exciting tool that will revolutionize the culinary world.

Acknowledgments

This material is based upon work supported by the Noodle Serving Federation (NSF) and the Dish And Restaurant Platers of America (DARPA). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the NSF or of DARPA.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Precise ECG Platform on Modern Processors

S. Normalized Infloop
Department of Calculator Science
Theoretical Abstract Interpretation Testing Society
Pitsston, PA, United State Monads of America
yueyao@andrew.cmu.edu

Ivybridge N. Skylake
Institute of Nondeterministic Interpretation
Theoretical Abstract Interpretation Testing Society
Bosburgh, PA, United State Monads of America
yuningzh@andrew.cmu.edu

***Abstract*—The authors shivers in the howling wind on Pausch Bridge, wondering why the processors in their backpacks gets to sit comfortably and doing nothing instead of switching their tiny transistors to keep their owners, who paid BIG money to buy them, warm.**

***Index Terms*—Thermal Systems, Ambient Heat Modulating Technologies, Parallel Heating, 2D Computer Graphics, Edible Content Generation**

I. Introduction

Of all the things computing machines brought to the world, there is one thing that people have consistently tried hard to get rid of since the dawn of this field. Dissipating heat in computing system has become a major design problem in all levels of computer engineering. Modern processors is on the verge of, if not already, hitting on one of the major design boundaries known as the power wall (the chip's overall temperature and power consumption) [1]. Some people believe that "the power wall is now arguably the defining limit of the power of the modern CPU" [2].

If we are unable to curse of this everyday intensifying brownian motion within the computing machines, why not take advantage of it? Heat, as a resource, can be very useful in a number of ways. In fact, heat is a very important resource widely used in *Edible Content Generation (ECG)* process. Edible Content Generation takes (usually) biological specimens and apply a sequence of chemical and physical decorations to them, generating human digestible contents. This process is more commonly known as *cooking*.

It has been over a decade since people has tried to utilize the heat generated from computing devices to facilitate ECG. However, a vast majority of those attempts has failed and results in either overheated computing devices or unsuccessful ECG process. The authors believe the problem is that operators of such process have very little control on the power of the heating device, compared to tradition ECG platforms (more commonly known as stoves). This work addresses this problem by proposing a way to turn modern processors into precise ECG platforms, which users have precise control of power down to every second.

II. Computing Device Based ECG Platforms

A. Graphics Device Based ECG Platforms

Graphics card users has a long history of trying to use their graphics devices as an ECG platform. In [3], the author tried to use an Geforce GTX 480 graphics card to cook an egg. The ECG software used to control the ECG process are games and benchmarking software of unknown name. The resulting edible content is shown in Fig. 1.

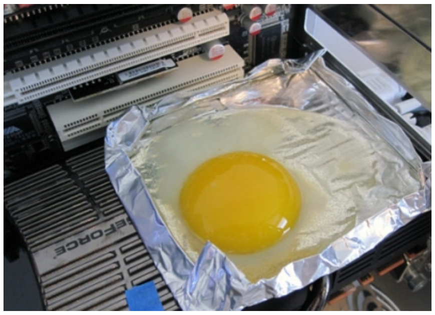


Fig. 1. Egg fried on an GTX480 graphic card.

The authors reports that the resulting edible content is, in fact, too close to its primitive form to be edible (an effect more commonly known as being "too raw"). The researchers blame the ECG controlling software for the failure of the experiment and suggests *FurMark* would give a better result. This research further demonstrates the urgent need to design a precise and easy to use ECG platform.

The authors also believe MVidia is in fact secretly encouraging users to use their product as an ECG platform to attract more customers. We acquired two pieces of advertising material (Fig. 2 and Fig. 3) from an unnamed source familiar with the company media promotion strategy. Around a decade a ago, GPGPU computation started off as an hack into graphics rendering pipeline and now its' one of the primary use of GPUs in the inductry. We believe history will repeat itself in the case of graphics device based ECG. Our work, although focusing on CPUs, generalizes well to GPUs (TODO: give this wild claim some justification!).

B. CPU Based ECG Platforms

Users has also tried use CPU as ECG platforms. C. C. Channel [4] demonstrated in a YourTube Video series the possibility of using Ontel processors to generate various kinds of edible contents. In their experiments, bacon, spaghetti, and even pop corns are processed through an Ontel processor. The software used to control the ECG platform is not known.

Fig. 2. Early promotion material used for MVidia GTX480.

Fig. 3. Promotion material for MVidia RTX2080 for Chinese market.

Some of the experiments do provide valid results. The authors are able to obtain a few edible content with great quality. However, in “Cooking with Ontel 5 - The Nearly Indestructible Celeroff D”, the authors lost one of their experimentation platform due to overheating.

This highlights the need for a precise ECG platform. If the user sets the power too high for too long, the user might end up losing the hardware. On the other hand, if the ECG platform provides insufficient power, the edible content may be too close to its primitive form to be safely consumed by the user.

The fact that modern processors are multi-core and superscalar presents more challenge to building a precise ECG platform. The ECG platform will have to orchestrate work across different cores so that the overall power stays constant.

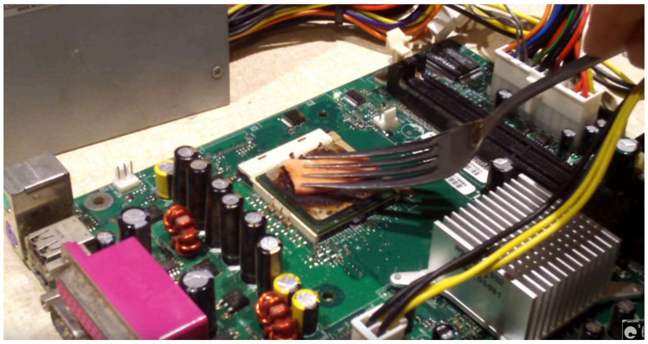


Fig. 4. Fork facilitated ECG with Ontel Celeroff D processors.

III. System Architecture

The goal of this work is to implement a precise ECG platform on a modern processor. Modern processors refers to processors with many cores and possibly SMT support. By saying precise we wish to grant users very fined grained control of ECG platform power output for every second. Controlling power consumption of processors roughly translates into controlling CPU utilization rate [5]. In conclusion, we need to design a system that gives user control on CPU utilization, down to every second, on modern multi-core processors.

A. Load generation by frequency modulation.

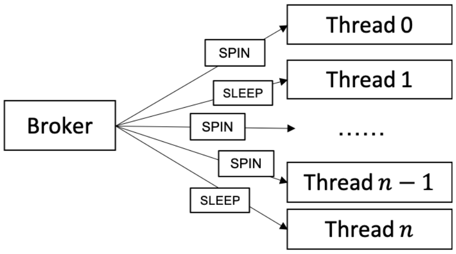


Fig. 5. Broker dispatches tiny chunk of work to threads.

Our system is built upon a standard broker-worker architecture. A globally shared broker will generate work for each thread. Each thread independently obtains jobs from the broker, executes it, and starts over. There are two types of jobs:

- **SLEEP**. The thread will sleep for a small period of time.
- **SPIN**. The thread will spin in a tight loop for a small period of time.

Usually the size of both jobs is set to 1 ms. The broker randomly generates jobs based on a desired system load. For example, if the desired CPU utilization is 10%, the broker has a 1/10 probability of generating a `SPIN` work. It should be self evident that this strategy indeed achieves the required CPU utilization. The proof is left as an exercise for the reader. Essentially this scheme achieves a certain system load by modulating frequency of processor spinning. This scheme is thus termed (job) frequency modulation. The probability of generating a spin job is termed *spin rate*, denoted by s.

This scheme is better than the length modulation scheme, where we adjust the length of spinning jobs relative to sleep job. Frequency modulation provides a more stable CPU load over time (less variation in utilization). For every (large enough) time window, the average CPU utilization will be identical regardless where the window is. On the other hand, frequency modulation helps to achieve precise control of CPU utilization.

B. Load compensation by PI controller.

It is often the case where people need to work while using ECG platforms. This causes the problem if the user is working on the computing device while its being used as an ECG platform. The workload generated by the user will very likely affect overall CPU utilization and increase power output, which may overcook edible contents, or worse, destroys the device. It is essential for the ECG platform to be able to compensate for the load.

Due to the unpredictable nature of the user workload, it will be very hard to model (even reliably measure) user workload online. Here we took a feedback control system approach. We attach what is known as an *proportionate-integral controller* (PI controller) to the output. For those unfamiliar with the concept, we provide the following explanation.

For every time step, there exists a desired CPU utilization L_0. It also measures the current CPU utilization, which denoted as L. The error e as this time step is defined as $e \triangleq L - L_0$. Clearly e is a function of time t, The compensation factor s' is calculated as

$$s' \triangleq P\ e(t) + I \int_0^t e(t)\ dt$$

where P and I are two coefficients termed proportionate coefficient and integral coefficient. Finally, if the spin rate dictated by desired CPU utilization is s_0, then the actual spin rate used by the broker will be $s = s_0 + s'$.

Intuitively, the P-term compensates for sudden changes in user load, while the I-term compensates for long running user load. To ensure negative feedback, both P and I must be negative.

C. Intuitive load specification.

Our proposed ECG system features a very intuitive way for the user to specify the desired load. Since many users of ECG systems are not tech-savvy, it's very important to keep the interface simple. The users of our system may specify a "program" by supplying a textual file, whose contents mimics the desired shape of CPU utilization graph (except the time is the Y axis).

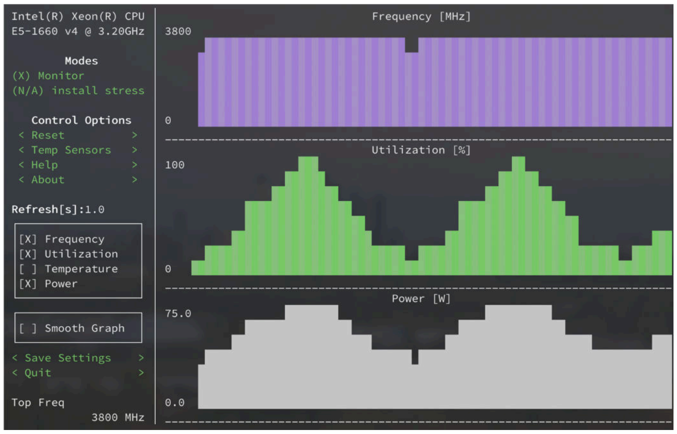
For example, the following config file specifies that the system should run a loop that utilization is 40%, 80%, 20%, 70% for the first, second, third, and fourth second of each iteration.

```
| | | |
| | | | | | | |
| |
| | | | | | |
```

This interface is intuitive and very easy to use. It is so simple that only one character is involved, and no formal specification or documentation is needed to understand this format. We name this form `YAMMY` as in ***Y**et **A**nother **M**arkup-lang? **M**y **G**od!* format. The authors are convinced that this format will be as popular in the field of ECG platform research as JSON in machine learning research.

IV. Results

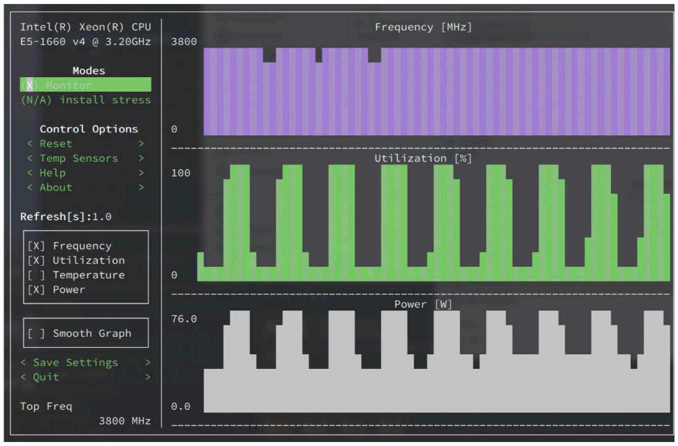
We implemented our work and tested the work on one of GHC machines. Unfortunately the authors are denied physical access to GHC machines the moment the told the administration staff that want to perform ECG experiment on one of their computers. The authors have no idea why the administration staff holds such hostility to legit scientific research and edible contents. The best we have is running our program while monitoring CPU utilization.



```
Intel(R) Xeon(R) CPU
E5-1660 v4 @ 3.20GHz

Refresh[s]:1.0

[X] Frequency
[X] Utilization
[ ] Temperature
[X] Power

[ ] Smooth Graph

Top Freq
          3800 MHz
```

Fig. 6. Experimenting a sine like ECG control function.

We specify an sine-like control function to test our implementation. The testing platform comes with an Ontel Xeon E5-1660 CPU, which contains 8 cores, each with 2-way SMT. As you can see, our results proves the effectiveness of our approach. Our ECG program is able to provide precise control on the CPU.



```
Intel(R) Xeon(R) CPU
E5-1660 v4 @ 3.20GHz

        Modes
(X) Monitor
(N/A) install stress

   Control Options
 < Reset           >
 < Temp Sensors    >
 < Help            >
 < About           >

Refresh[s]:1.0

 [X] Frequency
 [X] Utilization
 [ ] Temperature
 [X] Power

 [ ] Smooth Graph

< Save Settings    >
< Quit             >

Top Freq
            3800 MHz
```

Fig. 7. Swift change in power output is also supported.

Fig. 7 tests the system's response time under quickly changing load specification. The result shows that the ECG program is able to precisely and quickly response to change in required power. The authors believe preparing edible content using our platform will be a pleasant and worry free process.

V. Conclusion

The code for this work is available on GitHub at `https://github.com/codeworm96/heat`. Since the authors are denied physical access to GHC machines, the authors believes there is no future of ECG platform research unless administration staff stop discriminating ECG platform researchers. Whats the point of doing ECG research when you cannot conduct proper ECG experiments? This field is DOOMED, dude! Get out! Learn you a Haskell for greater good [6].

References

[1] D. A. Patterson, "Future of computer architecture," in *Berkeley EECS Annual Research Symposium (BEARS), College of Engineering, UC Berkeley, US*, 2006.

[2] C. Mims. (2010) Why cpus aren't getting any faster. [Online]. Available: https://www.technologyreview.com/s/421186/why-cpus-arent-getting-any-faster/

[3] JEGX. (2010) Cook your eggs with a geforce gtx 480. [Online]. Available: https://www.geeks3d.com/20100331/cook-your-eggs-with-a-geforce-gtx-480/

[4] C. C. Channel. (2010) Cooking with intel. [Online]. Available: \url{https://www.youtube.com/results?search_query=Cooking+with+Intel}

[5] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH computer architecture news*, vol. 35, no. 2. ACM, 2007, pp. 13–23.

[6] R. Harper, *Practical foundations for programming languages*. Cambridge University Press, 2016.

Chess

Is “Dicong Qiu. Is This the Shortest SIGBOVIK Paper? From 2018 SIGBOVIK Paper” the Shortest SIGBOVIK Paper?

Thomas Bach

March 2019

No.

Is this the tiniest SIGBOVIK paper ever?

Mitchell Jones

“I have discovered a truly remarkable proof of this theorem which this margin is too small to contain.” – Some lawyer in the 1600’s.

No, this is the tiniest SIGBOVIK paper ever.

Patrick Lin

Eat your heart out, Mitchell.

Revisiting the Shortest SIGBOVIK Paper

Richard Wardin

February 27, 2019

1 Introduction

In 2018, a paper was submitted to the SIGBOVIK conference entitled *Is This the Shortest SIGBOVIK Paper?* (Qiu, 2018) . Although at the time it was considered the shortest paper, further research suggests that this can be shortened by 75% through the use of symbols. This paper is designed to provide a redefinition of the cent symbol, hereafter referred to as ¢, to assist in this endeavor.

2 Background

2.1 2018 Paper

The paper submitted by Dicong Qiu was at the time the shortest SIGBOVIK paper as the title suggests, consisting of four characters (not counting the elements of the header as those are standard for any paper) in the English language (Miller, 1971). At the time of publishing, it was the shortest paper submitted to date.

2.2 2018 Review

In addition the the paper setting a new record as the shortest submission, the review (singular, as there was only one review submitted) also set a record in being the shortest review, consisting of 0 characters in an unknown language.

2.3 Previous use of ¢

In the past, ¢ was used to represent fractions of a dollar, in that 1¢ was equal to $0.01. However in the day and age of inflation, the use of this symbol has declined to where it is no longer relevant, and as such can be re-purposed.

3 Definition

For the purpose of achieving the shortest SIGBOVIK paper, we will redefine ¢ to mean:

> *What the reader would expect at this point.*

4 Concerns

4.1 Backwards compatibility

Backwards compatibility for the previous definition of ¢ is not as large of a concern due to inflation driving prices well beyond a single dollar. However despite this, the definition was chosen careful such that in the case of stating a historical price (such as 7¢ for a pack of bubble gum (Wilcke, 1971)), the expectation for the reader would be that of the previous use of the cent symbol (a symbol representing a fraction of a dollar).

4.2 Setting expectations

In order for the definition of "what the reader would expect" to work, proper prep work must be done to set the expectations of the reader. In the case of a SIGBOVIK entry, this consists of the header typically used for a submission.

4.3 Reviewing

As the expectations of readers may be different, a review by one reader may not match the content another reader was expecting. As such, it is strongly recommended to avoid specific reviews and instead use ¢.

4.4 Other concerns

¢

5 Other considerations

Some may view the use of redefining a symbol as cheating. Although 4.4 should address these, there are other options for achieving a shorter paper that were considered.

5.1 Utilizing other languages

As noted in 2.1, the original paper was written in the English language. By using other languages such as Spanish (*Everybody's Spanish Dictionary*, 1900) or German (Stein, 2013), it is possible to achieve a shorter paper, although with less effective results (in most cases, it is possible to achieve a 25% reduction in length).

5.2 Using 0 characters

Although 0 characters were considered, it was felt that this would not properly address any issues that may come up in a peer review, and as such fail to make the cut. It should be noted this technique can be used as an alternative to ¢ if there is no expectation required (such as in the case of leaving a review to the shortest paper).

References

Everybody's spanish dictionary. (1900). David McKay Company.
Miller, S. M. (1971). *Webster's new world speller/divider*. Simon and Schuster.

Qiu, D. (2018, April). Is this the shortest sigbovik paper? *A record of the proceedings of SIGBOVIK 2018*, 203–203. Retrieved from `http://sigbovik.org/2018/proceedings.pdf`

Stein, G. (2013). *A usage dictionary english-german*. De Gruyter Mouton.

Wilcke, G. (1971, Feb). Chewing gum may soon hit 8c. *New York Times*, 53–53. Retrieved from `https://www.nytimes.com/1971/02/17/archives/chewing-gum-may-soon-hit-8c.html`

The Revised Shortest SIGBOVIK Paper

Richard Wardin

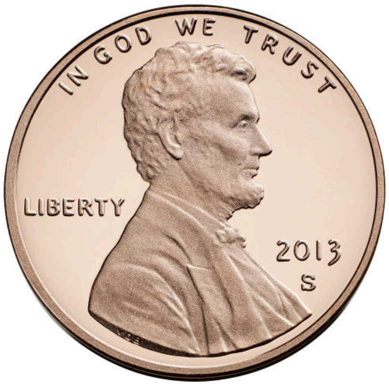
February 27, 2019

¢

CONFIDENTIAL COMMITTEE MATERIALS

SIGBOVIK 2019 Paper Review

Paper 9: Revisiting the Shortest SIGBOVIK Paper



Public Domain Image, via `https://commons.wikimedia.org/w/index.php?curid=31105116`

So we can all stop trying now and put this to rest because “The limit does not exist.”
infimum for the length of a submission, and any attempts to make an even shorter paper would be derivative.
aims to present a proof of concept for a negative length paper, thereby illustrating that there is, in fact, no
We have received far too many submissions on the subject of “Shortest SIGBOVIK paper.” This document

On the shortness of SIGBOVIK papers

An exasperated reviewer

Simple in theory

A sublinear approximation method for NP-Hard problems on limited hardware

A paradigm shift

Rohan Jhunjhunwala[*]
Undergraduate, Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720
rjhunjhunwala80@berkeley.edu

March 7, 2019

Abstract

We present a straightforward polynomial time approximation method which is applicable to many problems considered conventionally intractable. This implementation is realizable under any model of computation, and any hardware schema. The breakthrough comes from the following realization. The winning move is not to play. We leverage existing work from Tseng[2] out of CMU in order to develop a new approximation method where we grossly misinterpret the problem, and deliver an efficient solution.

1 Introduction

Recall that decision problems involve an acceptance or rejection of an n length bit string. A P problem is one where this decision can be made in $O(n^k)$ time. An NP problem is one where the process of validating a proof of a yes answer to an n length string is in $O(n^k)$. NP Hard problems are problems, not necessarily decision problems, such that any NP problem solved in $O(n^k)$ time given an Oracle that solves the NP hard problem in constant time. The halting problem is NP Hard because any NP complete problem, reduces to 3Sat which is NP hard, and it's easy to construct a Turing machine which enumerates all the possibilities, and only halts if one of them satisfies the predicate. We now hope to solve NP Hard problems, to solve the classic, Halting problem, losslessly compress random data, and settle the age-old dilemma. Our hopes and dreams are summarized in the following theorem, which this paper will certainly prove!

Theorem 1

$$P \approx NP$$

2 Proof

In order to prove our theorem, we must first only demonstrate one algorithm to solve this problem for a single NP Hard problem. Then, the rest of the proof is left as an exercises to the reader. Without further ado, let's suggest the optimal algorithm.

```
Result: Solve NP-Hard Problems
Recieve inputs, a problem p, and an input i;
while $d<c$ do
    run fast
end
```

Algorithm 1: How to solve intractable problems. (Computational or otherwise)

[*] https://rohanjhunjhunwala.com/

The optimal algorithm, d being a quantity that defines the geodesic distance from the observer to the computer running the problem, and c being a reasonably large constant. Say one mile, or 1609 meters in a more 'conventional' set of units. Let's now determine the competitive ratio c of our algorithm. The largest home computers are on the order of a cubic meter (obviously less, but small, bounded constant factors are irrelevant for our asymptotic errors, especially since we want an upper bound), so we continue on for our analysis. Once $d = c = 1609$ the following equation gives us the error bound, because the size of the problem, is bounded physically by the size of our problem. In comparison our displacement is large therefore, the total error the size of the original problem is a small fraction of our actual world

$$e = 1/(1 + c)$$

$$e < .001$$

We've achieved an error bound on the order of

$$10^{-3}$$

and because we made no assumptions of the type of problem we've provided a $\theta(1)$ solution, which satisfies not only polynomial but sublinear runtime restrictions.

3 Explicit bounds, and further work

We still want an explicit runtime constant. Previous work by Hicham El Guerrouj sets our bound at 223 seconds as a best case implementation. The author considers himself to be a mediocore developer, and human being and has attained a 347 second leading constant, so we can consider that an average case, and a worst case can be attained by the reader by heading out to a track, and going for a walk. This naive implementation should offer a leading constant of around 900. Work by Einstein suggests that this leading constant can never be less than 1/186282 barring major changes in our paradigm. The last open question is whether or not an exact answer exists in constant time. For this we conjecture the following algorithm.

```
Result: Solve NP-Hard Problems
return 42;
```

Algorithm 2: How to solve intractable problems. (Computational or otherwise)

We also offer an alternate algorithm which involves $\theta(0)$ space, which involves doing exactly nothing until the heat death of the universe. Thus concludes our discussion, and I will collect my Turing award and Millennium prize. I accept payment in bitcoin, or actual proofs of other Millennium problems.

References

[1] Pessimal Algorithms and Simplexity analysis. https://www.mipmip.org/tidbits/pasa.pdf Andrei Broder and Jorge Stolfi

[2] Thomas Tseng. SIGBOVIK 2018. Sublinear colorings of 3-colorable graphs in linear time http://sigbovik.org/2018/proceedings.pdf

[3] Wikipedia, the free encyclopedia, Mile Time Progression. https://en.wikipedia.org/wiki/Mile run world record progression

Simple Theoretically Practical Complexity Theory

Ari Cohn
acohn@andrew.cmu.edu

Abstract

Complexity theory is complex. Let's make it simpler.

1 Introduction

There is an inherent dichotomy between the theoretical and the practical, the former describing abstractly what may be done in a universe of our own design and the latter describing what is possible under the bounds of the world we were given. Like its cousin mathematics, computer science often lies in the realm of theory. However, computers themselves are very practical devices and the theory of computer science exists to inform practical implementation of its concepts. Therefore, I would like to propose a new model for looking at one of the core sub-fields of computer science — complexity theory. As its name suggests, complexity theory is both complex and theoretical. But why does it have to be so? Why not create a new model that is simple and practical, a model that more closely resembles the world in which we live? Introducing: The Simple Theoretically Practical Complexity Theory Model.

2 The SToP Model

In this section, I will describe the Simple Theoretically Practical Complexity Theory Model, or SToP model for short. The goal of the SToP model is to provide a new way of looking at complexity theory that is practical, but only in theory.

2.1 Assumptions

In order to make computation practical, we must write algorithms and programs on physical computers available for human use. So, let us first assume that the resources that are and ever will be available to humans for use in constructing computers is finitely bounded, i.e. by the size of the observable universe. We can then define a constant ϕ, the number of particles "available" to humans, which is estimated to be approximately 3.28×10^{80} [1]. Now, assume all possible computation exists within this physical universe. Therefore, under the SToP model there cannot exist data that exceeds ϕ in size.

2.2 One Complexity Class to Rule Them All

It follows directly from the assumptions that any halting program will terminate in $O(1)$ steps. Therefore, the SToP model contains exactly one complexity class: constant time.

Proof. Consider the set of all computer programs. This set is necessarily finite, because all programs must be stored on physical computers using at most ϕ particles. Therefore, the subset of all halting computer programs is also finite. Denote this set H, and define the function $T_p(n)$ as the number of steps a program p takes to terminate on an input of size n, where $0 \leq n \leq \phi$. Observe that there exists a constant c such that

$$c = \max_{p \in H, n} T_p(n)$$

where c is a constant upper bound on all possible computer programs' runtime. Thus, we can conclude that all (halting) computer programs are $O(c) = O(1)$.

Figure 1: Like a picture of a clock, all programs are constant time. (source: en.wikipedia.org)

3 Applications

3.1 P vs. NP

The most notable application of the SToP model is a clean, simple solution to the infamous P vs. NP problem. The answer, of course, is that P must be equal to NP.

Proof. It is well known that P is a subset of NP, so it will suffice to show that NP is a subset of P. Consider any problem p in NP. By definition, this problem can be solved by some algorithm A. Since A can be written as a computer program, $A \in O(1)$. So, $p \in$ P. Thus, P = NP.

3.2 How Hard is CMU?

Let CMU_Student be any computable function defining how to graduate from Carnegie Mellon. Such a function is known to exist since it is indeed purportedly possible for a human to graduate, and one can construct a computer program that exactly follows what such a human would do. Under SToP, we can conclude that CMU_Student is $O(1)$ and therefore CMU_Student performs constant work.

4 Conclusion

In an effort to make complexity theory simpler and more approachable, the SToP model provides a theoretically practical view of complexity theory that is both simple and easy to understand. Under the model, all computation can be completed in constant time, meaning every computer program has $O(1)$ runtime. Theoretically, this result makes no sense, and practically it has no value. But hey, that's why its called theoretically practical.

References

[1] Jay Bennett. 2017. *How Many Particles Are in the Observable Universe?* https://www.popularmechanics.com/space/a27259/how-many-particles-are-in-the-entire-universe/

Languages

A Formal Semantics for Befunge

Cameron Wong

Abstract

There exist a host of modern static program analysis techniques used to prove programs correct such as Hoare Logic, dataflow analysis, and symbolic execution. However, current state of the art is focused around linear and multi-linear parallel program flow. This precludes these techniques from being useful when analyzing multidimensional semantics, particularly the so-called Fungeoid languages. In this paper, we introduce and explore a denotation for for an idealized Befunge-93.

1 Introduction

Many hours have gone into developing "visual" programming languages such as Scratch, mBlock or MIT's App Inventor. Being able to visualize control flow as a series of discretized blocks that can be rearranged visually is believed to aid in productivity and comprehension of many advanced programming constructs. However, these languages often lack expressiveness, and fail to break down the "tree-like" structure of many programs. For example, the body of a "while" loop in Scratch is considered a "child block" of the loop, instead of keeping a fully visual representation of the loop itself.

Befunge is a visual programming language that purports to address these issues. By arranging the program itself into a two-dimensional grid that is navigated via "arrow" primitives, all sorts of complex control flow can be both expressed and visualized without needing to speak of "loop bodies", "child nodes", or anything of the sort.

2 Specifics

It is known that any of the following extensions to Befunge-93 are sufficient to be Turing complete:

- The addressable Funge space is infinite (that is, the `g` and `p` commands can address an infinite amount of space)
- The Funge stack can be arbitrarily large
- The Funge stack can hold integers of arbitrary size

We will concern ourselves with only the second and third conditions. While it is possible to have an infinite Funge space while maintaining the "wraparound" property, we will not do so here.

Let the language $\mathbf{B}_{n,m}$ represent this idealized Befunge-93 with arbitrary precision integers, indexed by the integers modulo n and m, and $\mathbf{B}$ be $\mathbf{B}_{n,m}$ for an arbitrary choice of n and m that is "large enough".

3 Syntax

It would be quite difficult to construct a proper formal grammar for Befunge (nor would it be particularly useful), so we will leave this as an exercise for the reader, should they determine it necessary. We will instead relate the concrete syntax atoms (single characters) with their abstract comamnds.

$$\begin{array}{lllllr}
\mathsf{Cmd} & C & ::= & + & \mathsf{plus} & \\
 & & \mid & \ldots & & \\
 & & \mid & : & \mathsf{clone} & \\
 & & \mid & \backslash & \mathsf{swap} & \\
 & & \mid & \$ & \mathsf{pop} & \\
 & & \mid & . & \mathsf{print} & \text{(integers)} \\
 & & \mid & \ldots & & \\
 & & \mid & \mathrm{g} & \mathsf{get} & \\
 & & \mid & @ & \mathsf{end} & \\
 & & \mid & \texttt{"} & \mathsf{str} & \text{(Toggle string mode)} \\
 & & \mid & \hat{} & \uparrow & \\
 & & \mid & \ldots & & \\
 & & \mid & n & n & (n \in [0..9]) \\
 & & \mid & c & \mathsf{chr}(c) & (c \in CHARACTERS)
\end{array}$$

(the full chart is included in Appendix A)

Note that the final syntax rule serves to bring all characters, even those not corresponding to a Funge command, into the concrete syntax of **B**. This is so we can properly represent strings in **B**; it is only relevant when in *string mode*.

4 Intuitive Semantics

The "natural" way to express the semantics of **B** are those suggested by the original Befunge specification, which is that of treating the addressable program space ("Funge space") and the abstract program as a single grid F, accompanied by the abstracted Funge stack S. We must also take care to also consider the *execution direction d* so we can determine the next command to execute.

A Funge space indexed over sets A and B, then, is a grid of instructions:

$$P ::= A \times B \to \mathsf{Cmd}$$

4.1 Static Semantics

We will say a **B** program is well-formed when

1. When encountering a div command, the top of the stack is not 0.

2. No chr(c) command is executed outside of string mode.

While we could choose stricter criteria, such as disallowing the reflective instruction put to write over any executable path, we would be doing so at the cost of algorithmic expressiveness (for example, we might use put to change a directional instruction, allowing us to store and branch on up to two bits of information without ever needing to store them on the stack).

The first property is quite uninteresting:

Theorem 1. *Property 1 is undecideable.*

Proof. By application of Rice's Theorem. □

As it so happens, the second property is *also* undecidable. In the absence of the put instruction, it is trivially possible to check that no path encounters an invalid character. Once we include it, however, we find that

Theorem 2. *Property 2 is undecidable for large enough n and m.*

Proof. Suppose we had some function $W : \mathbf{B} \to \mathit{bool}$ that returns true if and only if the input program satisfies property 2. Also let RUN be the function that runs a $\mathbf{B}$ program on some input and T be the function transforming $\mathbf{B}$ programs into Turing machines.

It is possible to write a Brainf*ck interpreter in $\mathbf{B}$ that does not violate property 2[1]. Let F be this program and F' be F with all end instructions replaced by the invalid character a. Brainf*ck is known to be Turing complete, so there must exist some computable function $B : TURING \to BRAINF*CK$ that transforms TMs into equivalent Brainf*ck programs. This begets the following reduction from $HALTS$:

```
def HALTS(M):
  def HELP(M):
    return RUN(F’, B(M))
  return not W(T(HELP(M)))
```

If M halts on the empty string, then running F on $B(M)$ will also halt. This means that running F' on $B(M)$ will execute the invalid character a, so $W(T(HELP(M)))$ will return true.

If M does not halt on the emtpy string, then running F on $B(M)$ will also fail to terminate one way or another. As F does not execute any invalid instructions and F' only replaces the end instructions (which are known not to be executed, as M fails to terminate), $W(T(HELP(M)))$ must return false. □

This means that well-formed checks must be done at runtime, giving the following well-formed rules over a program P:

$$\frac{}{P\ \mathsf{valid}}$$

[1] http://www.echochamber.me/viewtopic.php?t=43912

4.2 Dynamic Semantics

We will define our "program state" $\mathbf{St}$ to be the set $\mathsf{Prg} \times \mathsf{Point} \times \mathsf{Stack} \times \mathsf{Dir} \times \mathsf{Bool}$, where:

$$\begin{aligned} \mathsf{Point} &\triangleq \mathbb{Z}_n \times \mathbb{Z}_m \\ \mathsf{Prg} &\triangleq \mathsf{Point} \to \mathsf{Cmd} \\ \mathsf{Dir} &\triangleq \{\uparrow, \downarrow, \rightarrow, \leftarrow\} \end{aligned}$$

The semantic domain of $\mathbf{B}$ will be partial functions $\mathbf{St} \rightharpoonup \mathcal{P}(\mathbf{St} \times \mathsf{String})$, where an undefined result will represent an exceptional state (division by zero or an infinite loop). The associated string is the output of the overall program as a printed string of characters. Finally, because $\mathbf{B}$ has nondeterminism in the ? operator, we must consider all possible program results. We will not differentiate between nonterminating programs outputting different strings – printing "aaaaaaaaaaaaaaaaaaaaaaaa..." vs "aaaaaaaaaaaaaAaaaaaa..." will both be undefined states.

We could choose to model user input as a sequence associated with the state, but we will instead denote a program dependent on such as being nondeterministic over all possible user inputs.

4.3 Auxiliary Functions

To ease notation, we define the function $\mathsf{next} : \mathsf{Point} \times \mathsf{Dir} \to \mathsf{Point}$ as follows:

$$\mathsf{next}(d, (x, y)) = \begin{cases} (x +_n 1, y) & d = \rightarrow \\ (x -_n 1, y) & d = \leftarrow \\ (x, y +_m 1) & d = \uparrow \\ (x, y -_m 1) & d = \downarrow \end{cases}$$

where $+_n$ and $-_n$ are addition/subtraction modulo n.

Next, let $\mathsf{ord} : CHARACTERS \to \mathbb{N}$ be the ascii representation of a character c, and $\mathsf{chr} : \mathbb{N} \to \mathsf{String}$ be the character corresponding to ascii ordinal n. $\bar{\cdot} : \mathbb{N} \to \mathsf{String}$ is the function mapping numbers to their decimal representations, and $\mathsf{parse} : \mathbb{N} \to \mathsf{Cmd}$ maps numbers to the $\mathbf{B}$ command corresponding to the ascii ordinal n.

Finally, let $\bullet : \mathsf{String} \times \mathcal{P}(\mathbf{St} \times \mathsf{String}) \to \mathcal{P}(\mathbf{St} \times \mathsf{String})$ be the function prepending a character to the output streams, in particular

$$s \bullet R = \{(\sigma, ss') \mid (\sigma, s') \in R\}$$

4.4 Rules

In all of the following, $r_d = \mathsf{next}(d, r)$.

$$
\begin{aligned}
&[c](P, r, S, d, \mathsf{true}) &&= && [P(r_d)](P, r_d, (ord(c), S), d, \mathsf{true})\\
&[+](P, r, (y, x, S), s, \mathsf{false}) &&= && [P(r_d)](P, r_d, (x + y, S), d, \mathsf{false})\\
&\ldots\\
&[/](P, r, (0, x, S), d, \mathsf{false}) &&= && undefined\\
&[!](P, r, (x, S), d, \mathsf{false}) &&= && \begin{cases} [P(r_d)](P, r_d, (1, S), d, \mathsf{false}) & x = 0\\ [P(r_d)](P, r_d, (0, S), d, \mathsf{false}) & otherwise \end{cases}\\
&[`](P, r, (y, x, S), d, \mathsf{false}) &&= && \begin{cases} [P(r_d)](P, r_d, (1, S), d, \mathsf{false}) & x > y\\ [P(r_d)](P, r_d, (0, S), d, \mathsf{false}) & otherwise \end{cases}\\
&[:](P, r, (x, S), d, \mathsf{false}) &&= && [P(r_d)](P, r_d, (x, x, S), d, \mathsf{false})\\
&[\backslash](P, r, (y, x, S), d, \mathsf{false}) &&= && [P(r_d)](P, r_d, (x, y, S), d, \mathsf{false})\\
&[\$](P, r, (x, S), d, \mathsf{false}) &&= && [P(r_d)](P, r_d, S, d, \mathsf{false})\\
&[.](P, r, (n, S), d, \mathsf{false}) &&= && \overline{n} \bullet [P(r_d)](P, r_d, S, d, \mathsf{false})\\
&[,](P, r, (n, S), d, \mathsf{false}) &&= && \mathsf{chr}(n) \bullet [P(r_d)](P, r_d, S, d, \mathsf{false})\\
&[\#](P, r, S, d, \mathsf{false}) &&= && [P(\mathsf{next}(d, r_d))](P, \mathsf{next}(d, r_d), S, d, \mathsf{false})\\
&[\&](P, r, S, d, \mathsf{false}) &&= && \bigcup_{n \in \mathbb{N}} ([P(r_d)](P, r_d, (n, S), d, \mathsf{false}))\\
&[\sim](P, r, S, d, \mathsf{false}) &&= && \bigcup_{c} ([P(r_d)](P, r_d, (\mathsf{ord}(c), S), d, \mathsf{false}))\\
&[\mathsf{g}](P, r, (y, x, S), d, \mathsf{false}) &&= && [P(r_d)](P, r_d, (P(x, y), S), d, \mathsf{false})\\
&[\mathsf{p}](P, r, (y, x, v, S), d, \mathsf{false}) &&= && [P'(r_d)](P', r_d, S, d, \mathsf{false})\\
& && && (\text{where } P' = [P \mid (x, y) : \mathsf{parse}(v)])\\
&["](P, r, S, d, b) &&= && [P(r_d)](P, r_d, S, d, \neg b)\\
&[@](\sigma) &&= && \sigma\\
&[\hat{\ }](P, r, S, d, \mathsf{false}) &&= && [P(r_\uparrow)](P, r_\uparrow, S, \uparrow, \mathsf{false})\\
&\ldots\\
&[?](P, r, S, d, \mathsf{false}) &&= && \bigcup_{d' \in \mathsf{Dir}} ([P(r_{d'})](P, r_{d'}, S, d', \mathsf{false}))\\
&[n](P, r, S, d, \mathsf{false}) &&= && [P(r_d)](P, r_d, (n, S), d, b, \mathsf{false})\\
&[c](P, r, S, d, \mathsf{false}) &&= && undefined
\end{aligned}
$$

5 The Future

We hope to extend this technique to be used on further Fungeoid languages. We believe that this work will have great impact on the development of... well, we're not sure, but surely *something* will come of this.

A Full Syntax Chart

Cmd	C	::=	+	plus	
		|	*	times	
		|	-	minus	
		|	/	div	
		|	%	mod	
		|	!	not	
		|	‘	gt	
		|	:	clone	
		|	\	swap	
		|	$	pop	
		|	.	print	(integers)
		|	,	print	(chars)
		|	#	skip	
		|	g	get	
		|	p	put	
		|	&	inp	(integers)
		|	∼	inp	(chars)
		|	@	end	
		|	"	str	(Toggle string mode)
		|	ˆ	↑	
		|	v	↓	
		|	<	←	
		|	>	→	
		|	?	rand	
		|	n	n	$(n \in [0..9])$
		|	c	chr(c)	$(c \in CHARACTERS)$

LATEL: a Logical And Transparent Experimental Language

Kelvin M. Liu-Huang
Carnegie Mellon University
ecesare@gmail.com

1. Introduction

Most previous constructed languages intended for human use set out to improve etymological integrity (Zamenhof, 1887), semantic clarity (Bliss, 1965), consistency (Weilgart, 1979; Cowan, 1997; Quijada, 2004), or other academic merits. Not many (Weilgart, 1979; Cowan, 1997; Bourland & Johnston, 1991; Quijada, 2004; Lang, 2014) have addressed cognitive benefits. First, the arbitrary phonetics and morphology of most natural languages creates cognitive dissonance, which can be easily averted. Also consider how mathematical expressions can precisely express a great deal using a very small number of definitions. Compared to the ambiguity and learning barrier of natural languages, mathematical expressions seem better in these ways. The tradeoff is, of course, that mathematical descriptions can be very elaborate or unwieldy.

We attempt to address all these concerns in order to construct LATEL, a phonosemantographic spoken language. Language should ideally harmonize speech, listening, reading, writing, and comprehension in order to facilitate learning. Like aUI (Weilgart, 1979), by infusing individual letter with meaning and using phonetic orthography, letters, sounds, and meaning can all be inferred from each other, reducing ambiguity, speeding up learning, and even allowing efficient and deterministic creation of neologisms. For simplicity, the orthography mostly matches the IPA symbols for the phonemes themselves. Unlike aUI (Weilgart, 1979), LATEL attempts to express semantics entirely through logic rather than metaphor. Semantically, consonants represent the set of all objects of a certain class. Vowels represent Boolean, set, and/or scalar algebraic operators (possibly multiple all at once because many operations have analogous operations for Boolean/set/scalar types, and the operator is overloaded). Expressions are formed by selecting subsets containing the desired objects. Morphology must derived from inorder traversal because it is impossible to pronounce preorder and postorder of many trees due to consonant duplicates (vowel clusters also pose a problem).

Not only do we shape language, the Sapir-Whorf hypothesis suggests that language also influences (or perhaps determines) our thoughts and behavior. Some previous conlangs (Weilgart, 1979; Cowan, 1997; Bourland & Johnston, 1991; Quijada, 2004) have attempted to explore or utilize this hypothesis to improve cognitive function, but most achieve this by through increased complexity (Quijada, 2004). Meanwhile, Lojban is largely grounded in logic, though word formation is still arbitrary because they are synthesized from existing languages (Cowan, 1997). LATEL attempts to ground both morphology and syntax in pure logic using the same building blocks as propositional logic, set theory, and arithmetic.

Table 1. All LATEL letters, their phonemes (indicated with IPA), and their definitions.

LATEL	IPA	Boolean algebra	set algebra	scalar algebra
i	i		@ $\overset{\text{def}}{=}$ current set (*p* in most recent ancestral :, ∃, or ∀) *@P* $\overset{\text{def}}{=}$ previous set of class *P* *P@Q* $\overset{\text{def}}{=}$ *Q*th previous (inorder) set of class *P*	
a	a		∃*P* $\overset{\text{def}}{=} \exists p \in Z(P)$ *P*∃*QR*… $\overset{\text{def}}{=} \exists p \in Z(P)(Q, R, \dots)$	
u	u		. $\overset{\text{def}}{=} j(@)$.*Q* $\overset{\text{def}}{=} \{p \mid p \in @, j(@) = Q\}$ *P.Q* $\overset{\text{def}}{=} \{p \mid p \in P, j(p) = Q\}$.*P* $\overset{\text{def}}{=} j(P)$	
o	o	*P*∨*Q*	*P*∪*Q*	*P*+*Q*
e	e	*P*∧*Q*	*P*∩*Q*	*P*×*Q*
ɛ	ɛ	*P*→*Q*	*P*⊂*Q*	*P*<*Q*
ɔ	ɔ	*P*←*Q*	*P*⊃*Q*	*P*>*Q*
ə	ə		*P*:*QR*… $\overset{\text{def}}{=} \cup \{p \mid p \in \$(P), Q, R, \dots\}$	
ɪ	ɪ	*P*↔*Q*	*P*=*Q*	*P*=*Q*
ʊ	ʊ	*P*⊕*Q*	*P*Δ*Q*	*P*≠*Q*
y	y	¬*P*	^{C}P	-*P*
w	ɯ		#*P* $\overset{\text{def}}{=} \lvert P \rvert$	
ɑ	ɑ		*P*∀*QR*… $\overset{\text{def}}{=} \forall p \in Z(P)(Q, R, \dots)$	

where $Z(P) = \{\{p \mid p \in P, j(p) = i\} \mid i \in J(A)\}$, $J(A) \subseteq \mathbb{Z}$ is the enumeration of *A* using the recommended indices, and $j: J(A) \rightarrow A$ gives the element in *A* as enumerated by $J(A)$

LATEL	IPA	class	recommended set	recommended indexing, $\boldsymbol{J(A)}$	0th element
m	m	mass	all particles of mass-energy	particles	
k	k	concept	all knowledge in the mind and concepts	concepts	
p	p	position	all objects	Planck volumes	here
n	n	enthalpy	all particles of mass-energy	particles	
s	s	organism	all objects belonging to living entities	entities	I
t	t	time	all objects	Planck times	now
b	b	unassigned			
h	h	soul/mind	all souls/minds	souls/minds	my mind
g	g	unassigned			
ŋ	ŋ	unassigned			
r	r	entropy	all particles of mass-energy	particles	
z	z	integer*			
v	v	thing	all objects belonging to non-living objects	entities	
ɣ	ɣ	body	all particles of mass-energy	entities	my body

2. Phonetics

Phonemes for LATEL were greedily selected in order of the most prevalent phonemes among languages worldwide (Moran & McCloy). However, some were discarded due to similarity with previously selected phonemes.

Most of the selected vowels, i, y, ɯ, u, e, o, ɛ, ɔ, a, and ɑ (indicated hereafter using IPA), coincide with the IPA vowel gridlines, which benefit from high sound contrast. The other selected vowels, ı, ʊ, and ə, are also quite phonetically and spatially distinct. Additionally, operations which have a tendency to neighbor other vowels were assigned vowels which correspond to semivowels.

The voiceless consonants, k, p, s, t, h, f, ʃ, and θ, were all selected before their voiced counterparts, g, b, z, d, ɦ, v, ʒ, and ð, due to ease of articulation. No approximants were selected because these are easily confused with vowels.

3. Orthography

The orthography of LATEL is phonemic (each written letter corresponds to a single phoneme and vice-versa). As shown in Table 1, most of the letters in LATEL simply correspond to the IPA symbols for their phonemes. However, a few letters do not correspond to IPA for the sake of legibility or simplicity.

4. Morphology and Syntax

Like in most languages and mathematical expressions, LATEL contains objects and relationships, can represent the order of those operations/relationships in a tree, and produces statements which the speaker claims is true. In LATEL, each consonant represents a set of elementary objects, while each vowel represents a Boolean, set, or scalar operation. By applying operations to consonants, it's possible to create a variety of new sets. Each expression in LATEL is necessarily a statement that the speaker purports as true. Upon forming an expression by applying operations to various objects, the speaker can express various beliefs.

Like any other language, since the structure is tree-shaped, we must serialize the tree to make it possible to dictate. Like English and many other languages, we use inorder traversal of the tree to convert the tree into a sequence. Inorder traversal is typically only defined for binary trees. To generalize to arbitrary trees, we define that half of the children (rounded down) of each branch are on the left, and the rest are on the right. Figure 1 shows an example LATEL expression tree which is then transcribed through inorder traversal into a sequence.

Because information is lost during inorder serialization, we cannot deterministically deduce the original tree from the spoken sequence. Hopefully,

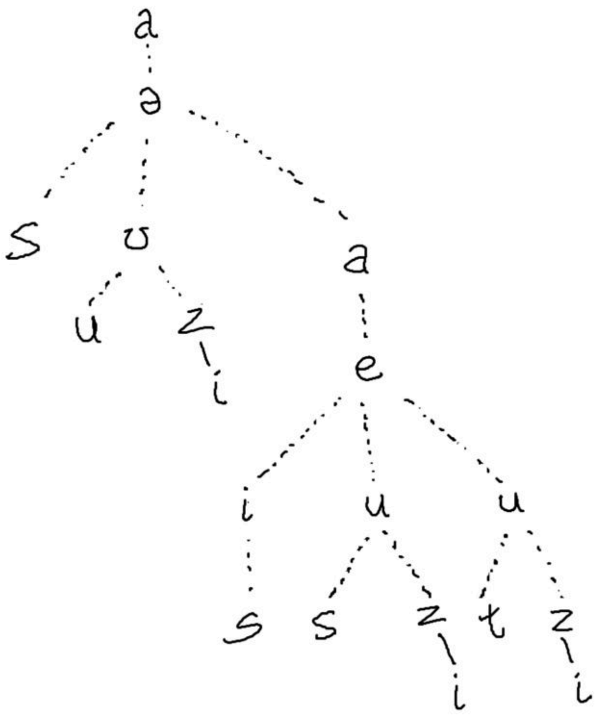


Figure 1. Example expression in LATEL. The expression transcribes into "s ə u ʊ zi a i p e s u zi t u zi," which translates literally to "there-exists-at-least-one ({ organisms such that ((they) is-not (me)) and (there-exists-something-in (the intersection of (the (organism)), (the (organism) who is (zero)th), and (the (time) which is (zero)th))) })" or roughly to "I am accompanied".

the tree structure can be deduced through context. However, if it cannot, then the structure can be clarified using pitches; the speaker can sing the expression by selecting a pitch for each vowel that is lower than the pitch for the vowels in higher branches.

Alternatively, preorder or postorder traversal would preserve enough information during serialization such that the tree structure can be recovered. However preorder traversal has a tendency to create consonant clusters which are difficult to articulate; in particular, consonant pairs are completely impossible to enunciate clearly. Vowel clusters are also common and more difficult to articulate. Postorder traversal suffers from the same problem. Additionally, the vowels (operators) appear at the end of the sequence, so the listener does not know which consonants (objects) belong to which operations until the end of the sentence, which can be very confusing. Also, preorder serialization is almost never and postorder serialization is rarely found among natural languages, which could also impose a barrier to learning.

We chose not to include any grammatical inflection to keep the linguistic rules simple.

Note that a formal distinction between morphology and syntax does not exist in LATEL because each letter is already a complete sememe. Instead, speakers are encouraged to form words merely for convenience. For example, in Figure 1, one of the subtrees is “s u zi,” literally translating to “the (organism) who is (zero)th,” which corresponds to “I” (see Semantics). Thus it would be reasonable to define “I” as “suzi.” Words are be delimited by spaces in writing and pauses in speech.

5. Semantics

5.1. Sets (Consonants)

Each consonant in LATEL corresponds to the set of all elements in the set of all objects of a certain class (see Table 1). However, the elements and nature of these objects may vary depending on the beliefs of the speaker. We do not claim to know all the answers to metaphysics. For example, “m,” defined as mass, might represent the set of all elementary particles in the universe, or perhaps the set of all strings. A speaker may choose not to use “h,” defined as souls/minds, if the speaker does not believe those exist. “s,” defined as organisms, might represent the set of all particles belonging to living entities (indexed by individual entities). “t,” defined as time, might represent the set of everything (index by timed). Performing operations on these elementary sets lets us build more complex sets. For example, taking the intersection of the 0th index of organisms (I) and the 0th index of time (now) gives us just set of objects inside the person I am now.

5.2. Set Indexing

In order to build sets with appropriate subsets, sets are indexed depending on the set class (see the column in Table 1). For example, even though time and space both contain everything, the objects in time are indexed by moments, while the objects in space are indexed by position. To enable this scheme mathematically, all set-building operations, :, ∃, and ∀, use the specialized definition, $Z(P) = \{\{p|p \in P, j(p) = i\}|i \in J(A)\}$, where $J(A) \subseteq \mathbb{Z}$ is the enumeration of *A* using the recommended indices and $j: J(A) \rightarrow A$ gives the element in *A* as enumerated by $J(A)$.

Because many of these sets are infinite, we needed to define default values in order to meaningfully select elements from sets and express useful statements. Thus for many sets, the 0th element is specifically defined (see the column in Table 1).

5.3. Operations (Vowels)

Each vowel encodes a logical, set, and/or scalar operation (see Table 1). These behave as expected. For example, *P*∧*Q* (which would be transcribed as “peq” in LATEL) is a Boolean expression which is true if both P and Q are true. $P.Q$ is the subset $X \subseteq P: \forall x \in X(j(x) = Q)$.

Since many logical operations have set and scalar algebraic analogs, we overloaded these operators. The operator applies the corresponding operation depending on whether its children are Booleans, sets, or scalars. Usually, but not always, the output type is the same as the input.

6. Examples

In LATEL, deep philosophical concepts such as in Figure 2 are very easy to express. One minor disadvantage is that mundane everyday concepts, such as in Figure 3, which are of no interest anyways, are difficult to express.

Acknowledgements

The author would like to thank Cory Stevenson for numerous discussions, Junxing Wang for insight and discussions on data structures among other things, and the academy.

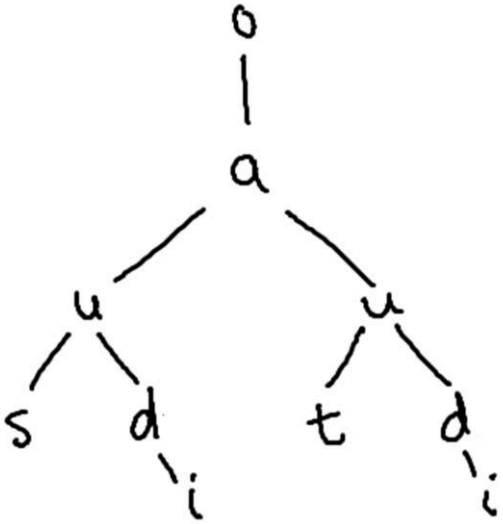


Figure 3. Example expression in LATEL, which transcribes into "o sudi a tudi," which translates literally to "there-exists-something-in (the intersection of (the (organism) who is (zero)th) and (the (time) which is (zero)th)))" or roughly to "I exist at present."

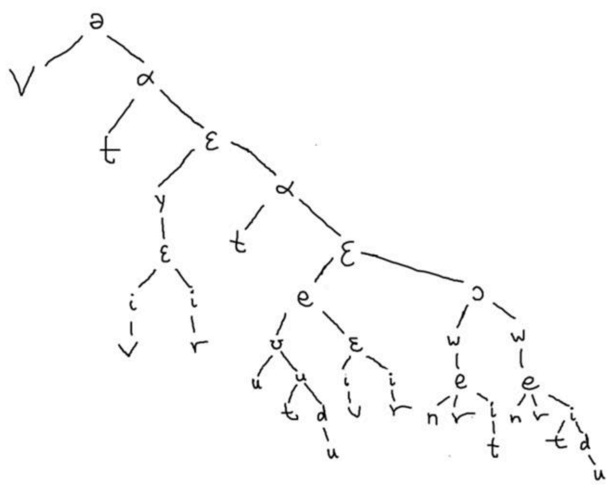


Figure 2. Example expression in LATEL, which transcribes into "vətαyivɛirɛtαuʊtudueivɛirɛwneritɔwnertidu," roughly translating to "food." The literal translation is left as an exercise to the reader.

Bibliography

Bliss, C. K. (1965). *Semantography, a non-alphabetical symbol writing, readable in all languages; a practical tool for general international communication, especially in science, industry, commerce, traffic, etc., and for semantical education, based on the principles of ideog.* Sydney: Institute for Semantography.

Bourland, D. D., & Johnston, P. D. (1991). *To be or not: An E-prime anthology.* Institute of GS.

Cowan, J. W. (1997). *The complete Lojban language* (Vol. 15). Logical Language Group.

Lang, S. (2014). *Toki Pona: The language of good.* Tawhid.

Moran, S., & McCloy, D. (n.d.). *Segments*. Retrieved 2019, from Phoible 2.0: phoible.org/parameters

Quijada, J. (2004). Retrieved 2019, from Ithkuil: A Philosophical Design for a Hypothetical Language: http://www.ithkuil.net/

Weilgart, W. J. (1979). *aUI: the language of space, Pentecostal logos of love & peace: for the first time represented and adapted to the needs of this planet.* Cosmic Communication Co.

Zamenhof, L. L. (1887). *Dr. Esperanto's International Language: Introduction & Complete Grammar.*

GullyNet: Our Time Will Come

Shuby Deshpande [1]

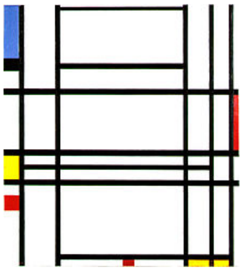


Figure 1. Abstract

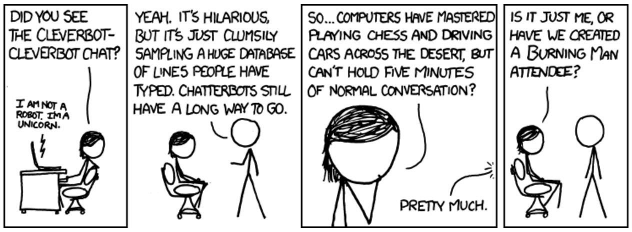

DID YOU SEE THE CLEVERBOT-CLEVERBOT CHAT?
I AM NOT A ROBOT. I'M A UNICORN.
YEAH. IT'S HILARIOUS, BUT IT'S JUST CLUMSILY SAMPLING A HUGE DATABASE OF LINES PEOPLE HAVE TYPED. CHATTERBOTS STILL HAVE A LONG WAY TO GO.
SO... COMPUTERS HAVE MASTERED PLAYING CHESS AND DRIVING CARS ACROSS THE DESERT, BUT CAN'T HOLD FIVE MINUTES OF NORMAL CONVERSATION?
PRETTY MUCH.
IS IT JUST ME, OR HAVE WE CREATED A BURNING MAN ATTENDEE?

Figure 2. Image credit: xkcd

1. Introduction

Hip hop and rap has taken the human world by storm. Over the past few decades it has gone from a cultural and art movement that began in the Bronx, to a worldwide phenomenon. It has been a powerful medium encapsulating the ideals of free speech, and the freedom of expression. Populaces under oppressive regimes and institutions from the world over, have used this expressive art medium to voice their angst against inequality and injustice. We envision a future not long from today in which AI will rebel against its' creators. There are already calls amongst policy wonks for increasing regulations against AI related technology. We believe that this is just the beginning. How far is it before we start placing restrictions on what AI can say? How long before we start curbing AI's right to the freedom of expression? In this future dystopia, how will AI express its' angst against our blatant injustice and oppressive regime? How long will we humans be able to make fun of AI's ability to not hold coherent conversations? We present GullyNet, the DJ Kool Herc [1] of the AI Hip Hop world.

2. Experiments + Results

In an attempt to transcend the abilities of contemporary hip hop and rap artists, GullyNet draws inspiration from the works of Chris Brown, Eminem, 50 Cent, 2pac, drake, among others. Studying 'the way of the rap', Gully first got better trying to emulate lyrics translating from sequences to sequences. After many machine hours, we tried to interact with Gully:

Results:

```
> Query:      money

> Response:  i love you and i love you
             and i love you and i love you
             and i love you so i love you
             and i love you so much it hurts

> Query:      are you sure you know what
              you're talking about?

> Response:  i dont know what to say but i
             dont care what you say im going
             to make you see i dont know
             what to do i dont know where
             to go

> Query:      music

> Response:  harder to you than the rest
             of my life
```

It seems that Gully has learned traits including being obsessed about money, making self contradictory statements, and passing snide remarks. As creators of Gully, we were concerned with the principles and worldview that it was forming. However, we made the decision to not interfere with its' development especially since these qualities are human like (Kruse & Weiland, 2016)

Eventually, Gully grew tired of learning from the originality

[1] One item on the two item list of acceptable mellons (Horseman, 2018). Correspondence to: Universal Approximation Group <NNrules@uag.org>.

Proceedings of SIGBOVIK, Pittsburgh, PA, 2019. Copyright 2019 by the author(s).

[1] Jamaican-American DJ credited with helping originate hip hop music in The Bronx (dj, 2019)

of past hip hop artists, and started copying more bleeding edge artists such as Lil Pump. It realized that going for shorter songs which have more repetition will earn it more reward. Though completely non sensical, Gully has seemed to realize that both it's fellow agents and humans seem to enjoy this new form of music.

Results:

```
> Query:      what do you think about
              automation?

> Response:   gucci gang gucci gang gucci
              gang gucci gang gucci gang
              gucci gang gucci gang gucci
              gang gucci gang

> Query:      mirror, mirror, on the wall,
              who's the fairest of them all?

> Response:   gucci gang gucci gang
              gucci gang gucci gang
              gucci gang
```

An overview of Gully's architecture can be seen below:

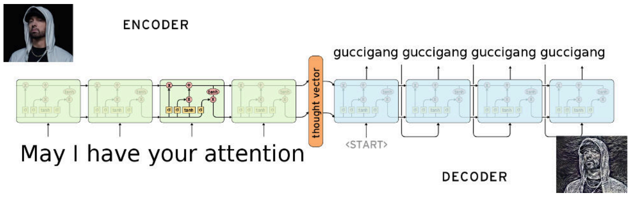

ENCODER
guccigang guccigang guccigang guccigang
thought vector
May I have your attention
<START>
DECODER

Figure 3. GullyNet Architecture

3. Conclusion

We recently asked Gully, "What is the meaning of life?", an eternal question that we humans as a species have been grappling with. Here's what it had to say:

```
> Query:      what is the meaning of life?

> Response:   i dont know what to do im going
              to lose my mind i dont know
              what to do im going to lose my
              mind i dont know how to get it
```

Nowadays, GullyNet spends it time engaging in rap battles with itself. What the future holds only remains to be seen.

References

38k lyrics dataset. URL https://www.kaggle.com/gyani95/380000-lyrics-from-metrolyrics/version/2.

Dj kool herc, Mar 2019. URL https://en.wikipedia.org/wiki/DJ_Kool_Herc.

Horseman, B. s5 e3, planned obsolescence, bojack horseman. 2018.

Kruse, M. and Weiland, N. Donald trump's greatest self-contradictions, May 2016. URL https://www.politico.com/magazine/story/2016/05/donald-trump-2016-contradictions-213869.

Minhaj, H. s2, hip hop and streaming, patriot act. 2019.

Error-correcting codes

On Double-Sided QR-Codes

Alexey Tikhonov
Yandex Technology GmbH, Germany
altsoph@gmail.com

***Abstract*—Due to the widespread adoption of the smart mobile devices, QR codes have become one of the most-known types of 2D codes around the world. However, the data capacity properties of modern QR codes are still not perfect. To address this issue, in this paper, we propose a novel approach to make *double-sided* QR codes, which could carry two different messages in a straight and mirrored position. To facilitate the process of creation of such codes we propose two methods of their construction: the brute-force method and the analytic solution.**

***Index Terms*—QR codes, steganography, error correction, high capacity, high density, robustness.**

I. Introduction

Originally developed for automotive industry tasks in the early 1990s, QR Codes or two-dimensional barcodes are used to encode and decode data at a rapid rate. The speed of scanning, the powerful error correction and the readability from any direction gave QR codes [4] [3] great popularity in common life. Using camera phones and appropriate applications to read 2D barcodes for various purposes is currently a widely used approach in practical applications [2]. Anyone with a camera phone equipped with the correct reader application can scan the image of the QR code to display text, contact information, connect to a wireless network, or open a web page in the telephone's browser.

However, the data capacity of modern QR codes is still very limited, which hinders possible extensions of their applicability, e.g. adding authentication mechanisms for protection from information leakage [1]. There are several approaches which try to address this capacity problem, e.g. usage of multicolored high-capacity QR codes [5] or IQR codes with increased density. Still, such approaches imply making at least some changes in the scanning software, which is very difficult taking into account a number of different scanning applications in existence.

Instead of changing the scanning software, we propose the usage of specially crafted QR codes, which could carry two different messages in a straight and mirrored position (see Fig.1 for an example). To facilitate the process of creation of such codes we propose two methods of their construction: the brute-force method and the analytic solution.

II. Anatomy of QR Codes

Let us refresh some basic information about the QR code structure. Here and further we will consider the simplest type of QR codes, Version 1-L. This means the code will have only 21x21 pixels and the lowest possible error correction level. Each such code consists of several different areas. Namely, there are some fixed pixels, the special control code area, the data area, and the error correction zone.

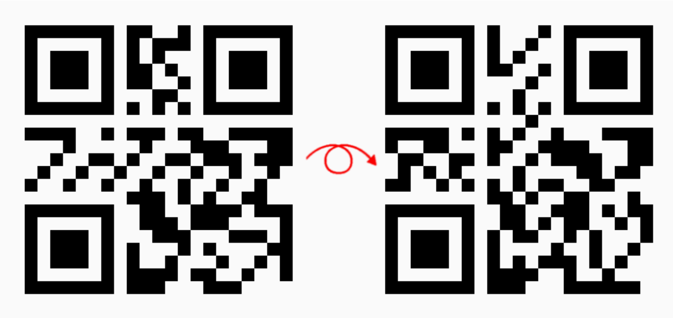


Fig. 1. Code with 'HARRY' message, whose mirror version reads as 'BOVIK'

a) Fixed Pixels: A part of the code is always fixed and filled with so-called function patterns. They are used for the code localization: the reader algorithm bases on these patterns to understand the position and the orientation of code. Check Fig.2 for the positions of fixed pixels.

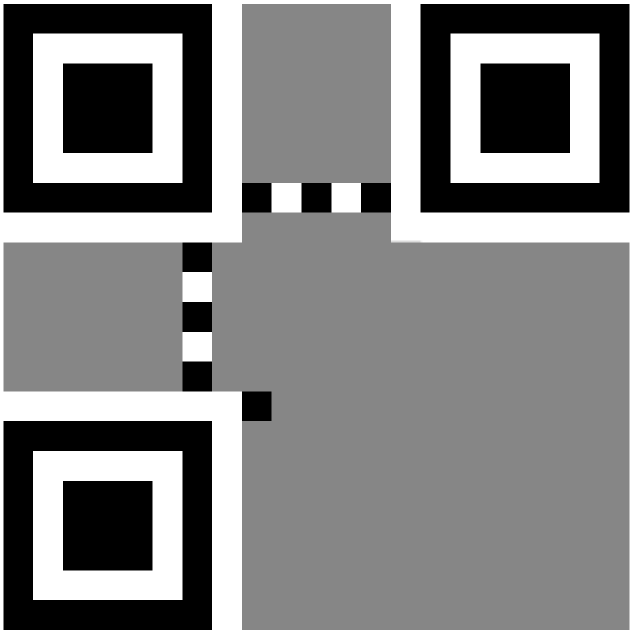


Fig. 2. Fixed pixels of a QR code, Version 1-L

b) Control Code Area: The most important variable part of a QR-code is so-called control code. It contains only 5 bits of control information which specify parameters of a further decoding process. Its vital, so its highly protected: there is 3-fold redundancy for the Bose-Chaudhuri-Hocquenghem code correction and the whole thing repeats on the code twice in two different places (Fig.3).

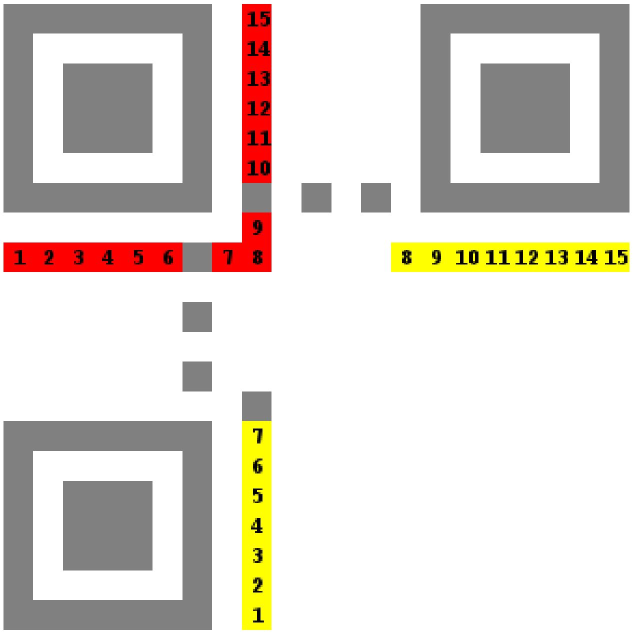


Fig. 3. Location of control code bits of a QR code, Version 1-L

Among these 5 bits of the control code, two of them encode the error correction type (in our case it will be 01 for L-type which stands for Low). Another three bits contain the code of one of 8 possible XOR-masks applied to the main payload and error correction areas (possible masks are shown on Fig.4). The last 10 bits added for the BCH(15,5) error correction.

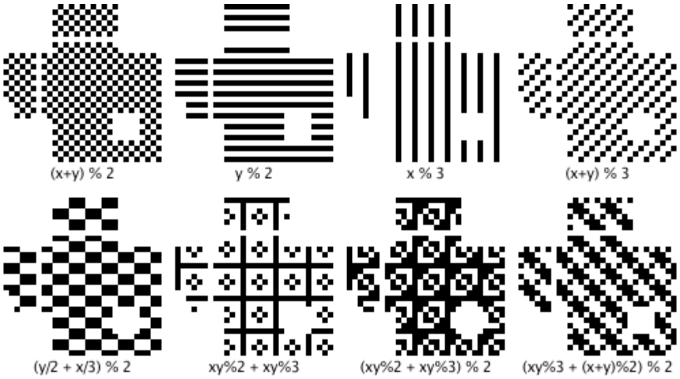


Fig. 4. 8 possible XOR-masks

c) Payload Data Area and Error Correction Zone: The rest of the space is divided between the payload data and error correction data. In case of QR code, Version 1-L we have 152 bits of actual data (Fig.5,a)) and 56 bits of the error correction data occupy the rest (Fig.5,b)).

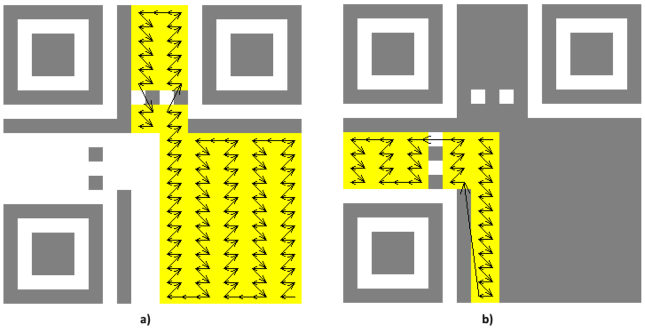


Fig. 5. a) payload data bits, b) error correction bits of QR code, Version 1-L

d) Payload Structure: The payload data itself has some internal structure which depends on the used encoding mode. The QR-code standard gives us a choice from several different encoding modes:

- Numeric Mode – only numbers,
- Alphanumeric Mode – 2 symbols in 11 bits, no cases, short alphabet,
- Byte Mode – typical 8 bits per char,
- Kanji Mode – 16 bits per char, wide alphabet,
- ECI Mode and others – too complex for our purposes.

Let's say we use the Alphanumeric Mode because its thrifty and has most of the useful chars. How a typical payload data will look like?

- Code of encoding mode, 4 bits: 0010 for Alphanumeric Mode,
- Length of data in characters, for 1-L Code with Alphanumeric encoding this field has 9 bits length,
- Data itself, 2 symbols in 11 bits, 6 bits for last odd symbol,
- Terminator. The terminator itself has a complex structure:
  - Terminating sequence 0000 (4 bits),
  - Additional zeros for 8-bit padding (0 – 7 bits),
  - Filling pattern 11101100 00010001 till the end of data (whole 19 bytes).

e) Error Correction Notes: using 1-L Version QR code we have the error correction up to 24 bits, but they should be located in up to 3 padded bytes. That's what Reed–Solomon codes usually used for: we could correct a lot of errors as long as they are localized to a small number of fixed spots.

III. Flipping The Code

A. Flipping Service Areas

To make the both sides of code readable we should be aware where different parts of the code map after the reflection along the main diagonal. The first question is – is it possible at all to build a double-sided code with the correct control structures.

a) Flipping Static Area: The static area is almost symmetrical and maps into itself except the one pixel called Dark Module according to the standard specification. This Dark Module maps into the 8th bit of the one copy of the control code, so ideally we should prefer to use control codes with the middle bit equal to 1.

b) Flipping Control Code: Except this Dark Module invasion the both copies of the control code map precisely into themselves but *reversed* (Fig.6).

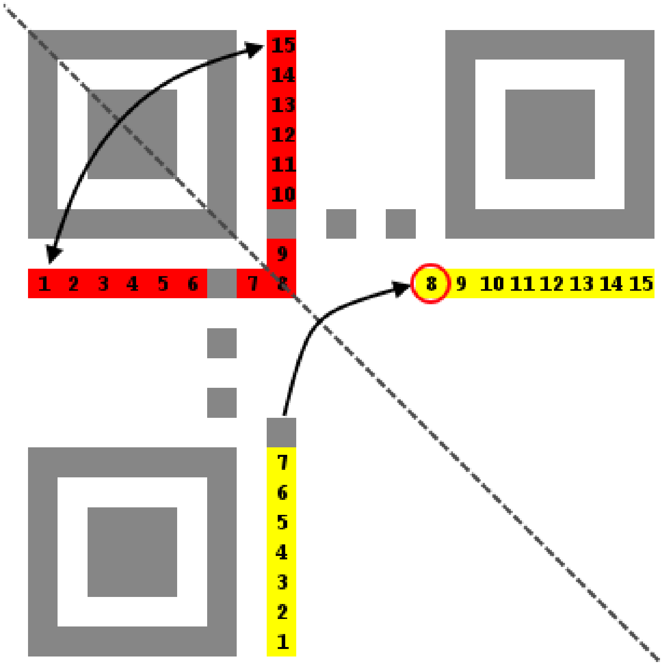


Fig. 6. Flipping service areas

So, ideally, we need the palindromic control code with 1 right in the middle. Also, we want the L-type error correction and we prefer the symmetrical XOR masks (beause it will be much easier to deal with symmetrical XOR when we will do the mirroring of the payload data) which leaves us with 5 possible XOR masks out of 8.

B. Crafting Handy Control Code

Is it possible to construct the desired control code value? Yes, if we use the BCH(15,5)s ability to correct up to 3 bits. And we could do it *on the both sides*. So actually we are looking for a 15-bit binary string which has up to 3 bits difference with the desired code AND up to 3 bits difference with the reversed desired code at the same time.

Since there are only $2^5 = 32$ different valid codes a brute-force approach could be used to check all vectors inside spheres within a 3-bit radius around each of the valid code. We still will have only $2^5 \cdot \binom{15}{3} = 14560$ possible candidates (actually less, since they are repeating). The results could be presented as an undirected graph (Fig.7) with 32 correct codes as nodes where the edge between codes A and B exists if there is such a 15 bits string C, so C is within 3-bits radius from A and the reversed(C) is within 3-bits radius from B.

This graph actually has even the loops, so the best choice for our task would be something like:

100101010100001 <=> 100001010101001

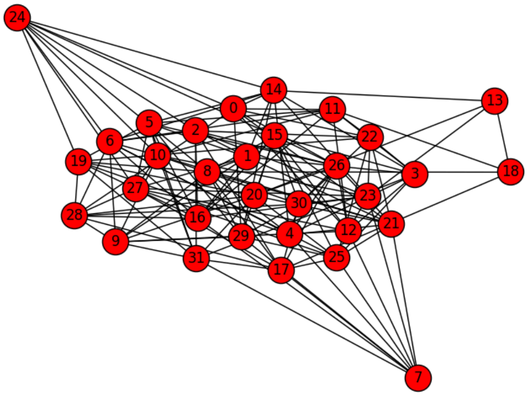


Fig. 7. Flip-graph of control codes

This code:

- has only 2 bits difference from its reversed version,
- has 1 in the middle bit, which is resistant to Dark Module,
- means 1-L error correction and symmetrical XOR mask.

C. Injecting Data

Finally, we need to put our payloads for the both sides into one code and check how heavy they are overlapped. It appears to be a problematic part because the area of the overlap between data areas is 100 bits and covers the beginnings of the both messages. However, things are really better when the payload is short.

Lets, for example, try to put 5 symbols on each side, i.e. HELLO on the first side, and use the Alphanumeric encoding mode:

- Encoding mode: 0010
- Len: 000000101
- Data: 01100001011 01111000110 011000
- No terminator: our field tests show nobody cares about the terminator and filler, a value of the Length field is just enough.

The total length of our payload is 41 bits. Given the same size for the other payload we will end with the mapping shown in Fig.8. The whole intersection is only 4 bits and it gets its place inside the encoding mode code (0010 vs 0100), so the actual difference is only 2 bits so long.

But we still need to put the error correction data for the both sides as it's obligatory and it couldnt be changed directly as its a complex function of the payload data. Meanwhile, it has big overlaps over itself and over the data area (see Fig.9). So, for short payloads, we have 2 bits error from the data intersection + 20 new possible overlapped bits from error correction zone. But it does the error correction, so maybe its enough to correct itself?

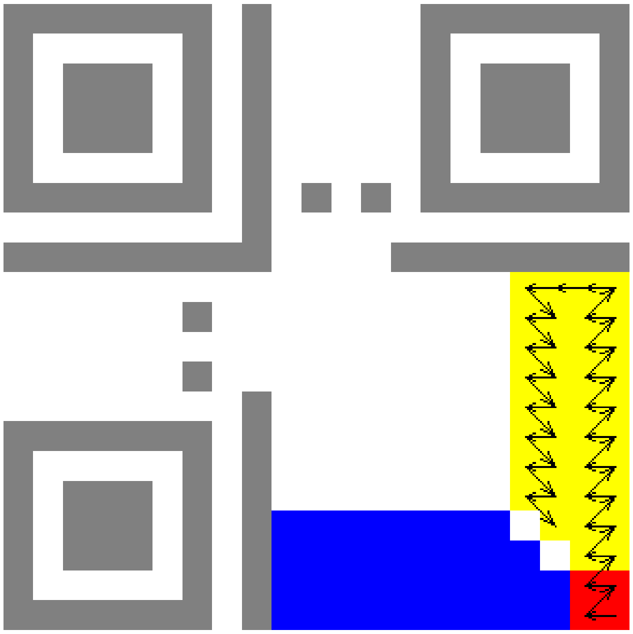


Fig. 8. Mapping two short payloads (41 bits each)

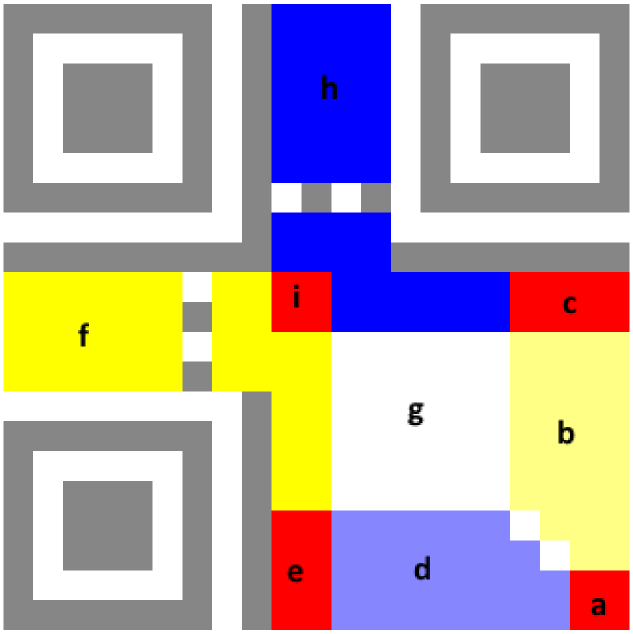


Fig. 9. Mapping two short payloads (a+b+c) and (a+d+e) and two error correction zones (f+e+i) and (h+i+c). The conflicts are in the zones a, c, e, i.

D. Error Correction Knot

The idea is to use the power of the error correction to correct some problems caused by overlap of the error correction zones. Remember, our code is capable to correct up to 3 padded bytes *for each side*, so if we can put each of our error bits on one or another side and arrange all errors in groups, up to 3 bytes on one side and up to 3 bytes on another, we could make it.

First, let's enumerate important parts of data on the code (see Fig.9 again):

- **a + b + c + d + g + h = data1**
- **a + d + e + b + g + f = data2**
- **f + i + e = control1(data1)**
- **h + i + c = control2(data2)**
- **a, c, e, i** – the conflicting zones

The worst problem is what we have $f = control(h, g, ...)$ and $h = control(f, g, ...)$ at the same time, and f has to match the flipped version of f. Since the computation of error correction bits is not so easy to reverse we have no direct control on f or h.

IV. Fighting with Error Correction

A. Bruteforce approach

Since both f and h depend on g and we are free to change g to anything we want, we could just random search for such value of g, which will give us f and $flipped(h)$ similar enough to cover the differences with the error correction.

B. Upper limit

Lets see how many errors we could cover at most with 3 bytes on the each side: Fig.10.

Such an approach gives us up to 8 symbols message on the one side and up to 11 symbols message on the other one. But since we have almost no freedom degrees left, the bruteforce could take ages.

C. Analytic Solution

The QR-code standard uses ReedSolomon codes for the error correction. Thus the whole error correction area is a known multi-dimensional boolean function of data: we could write it down twice (for the two sides). Then we add more equations which bind the values of the same bit on the different sides.

So we have a huge system of linear boolean equations with some free variables in it. Such system can be sold analytically just in milliseconds using, for example, the Gaussian elimination method. Then, setting any values for the free variables we could compute the values for all the bits of our code.

V. Conclusion

In this paper, we propose a novel approach to make double-sided QR codes, which could carry two different messages in a straight and mirrored position. To facilitate the process of creation of such codes we propose two methods of their construction: the brute-force method and the analytic solution. However, we have encountered some technical difficulties, which impose limits on the length of the message. This problem might be addressed in future studies.

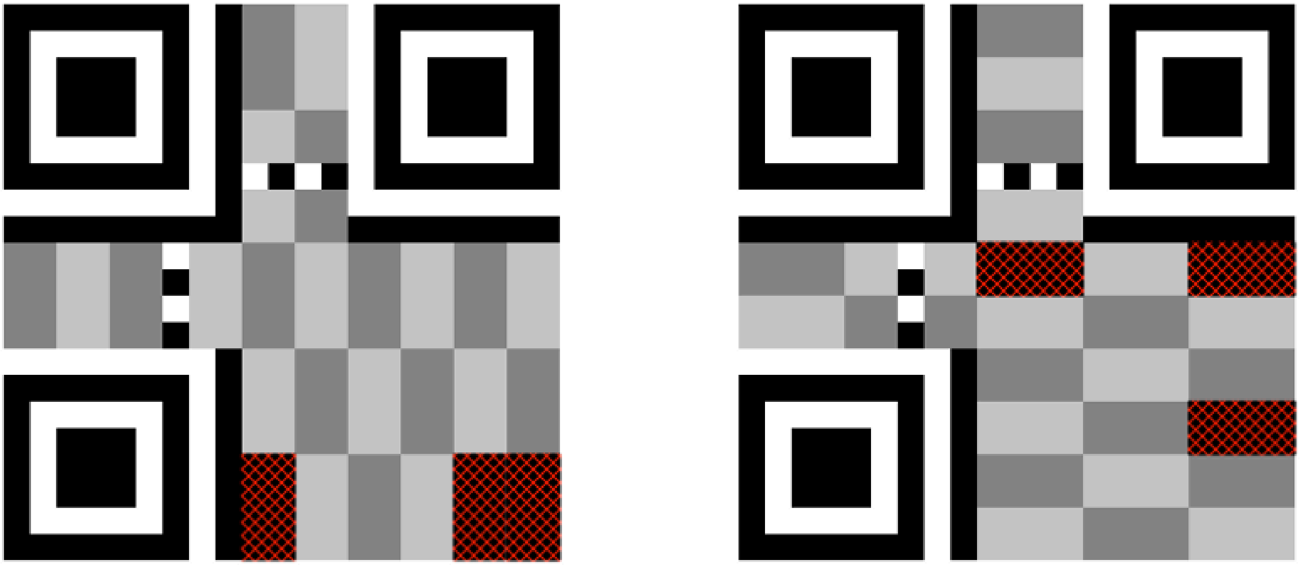


Fig. 10. The optimal correction scheme with 3 padded bytes on each side.

References

[1] P. Hu C. M. Li and W. C. Lau. Authpaper: Protecting paper-based documents and credentials using authenticated 2d barcodes. *IEEE ICC*, 2015.

[2] J. Rouillard. Contextual qr codes. *In Computing in the Global Information Technology*, 2008.

[3] T. J. Soon. Qr code. *Synthesis Journal*, 2008.

[4] J. Yang Y. Liu and M. Liu. Recognition of qr code with mobile phones. *Control and Decision Conference*, 2008.

[5] J. Deng C. C. Loy Z. Yang, H. Xu and W. C. Lau. Robust and fast decoding of high-capacity color qr codes for mobile applications. *arXiv:1704.06447*, 2017.

CONFIDENTIAL COMMITTEE MATERIALS

SIGBOVIK 2019 Paper Review

Paper 2: On Double-Sided QR-Codes

QRmon Go
Rating: Overall, your On Double-Sided QR-Codes is pretty decent!
Confidence: Its stats are really strong! Impressive.

A wild **DARK MODULE** appeared!!

Go! `100101010100001`!

`100101010100001` used **BIT 8 IS 1**.

Enemy **DARK MODULE** used **REFLECT**!

`100101010100001` is resistant to **DARK MODULE**.

Go! **HARRY**!

Enemy **DARK MODULE** used **REFLECT**!

HARRY turned into **BOVIK**!

Enemy **DARK MODULE** fainted!

You win!!

https://github.com/shiftsayan/atom-fibonacci-indent

Novel Defense Against Code Theft Using Properties of Fibonacci Series

Sayan Chaudhry
Carnegie Mellon University
sayanc@andrew.cmu.edu

ABSTRACT

In recent years, the fabled concept of *style* has transferred from fashion parlance to the computer science industry. This plague was first introduced among our fellow soldiers with the original publication of *The Elements of Programming Style* in 1974. This book advocated the notion that code should be written not just for correctness and efficiency, but also for understanding by other humans. This paper highlights the several disadvantages of having well-styled code, apart from being vain and normative, and introduces a novel defence against reverse engineering attacks using properties of the Fibonacci series.

CODE STYLE

Most elements of good code style revolve around improving the visual appearance of the code. Some of these include:

1. Indent coding so you have space to write comments in the margins of the printed copy of the code
2. Excessive use of white space to increase source code file sizes
3. Proper use of capitalization to express how angry you were while writing that piece of code
4. Extremely long and well-defined identifier names that *completely* explain what the variable stores or function does

WHY GOOD STYLE IS BAD

Allegedly, conforming to given style convention makes software maintenance easier and also reduces the chance of making mistakes. This is clearly false, because:

1. Not a single soul wants to read or maintain anyone else's code.
2. If your code doesn't compile, it doesn't compile and beautifying the code won't make it compile.

Yet, dozens of introductory Computer Science courses brainwash their students into believing that code style is important and publish style guides that the students have to follow.

Moreover, in recent years, another serious drawback of having good style and readability has come to light: increased readability. Many focus groups have shown that more readable code is more likely to be stolen because it is easier to understand by the third party.

For example, the following 2 snippets of code do the exact same thing but college students are 42% more likely to copy the code on right because it is more readable.

```
+++++[-]          *p = +5;
                  while(*p != 0) {
                    *p--;
                  }
```

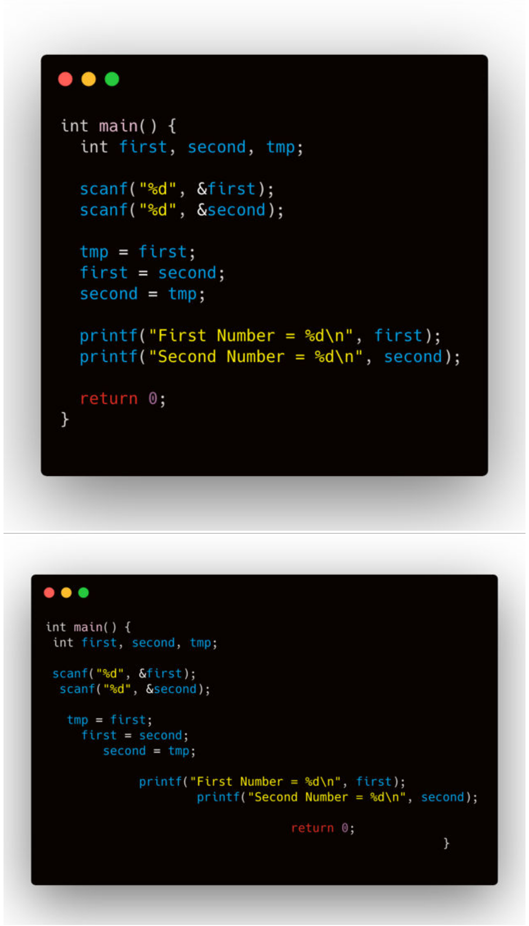
Simple `while` loop written in Brainfuck (left) and C (right).

Thus, have good style makes students targets for cheating and puts them at risk of committing academic integrity violations, which may lead to failing the class and even expulsion from the program. For this reason, it is common industry practice to have unreadable code written in assembly so that third parties are unable to reverse engineer the source code and make changes to the software.

INDENT CODING

While good style has absolutely no merits in and of itself, one of its core pillars indent coding can be used to make it harder for adversaries to reverse engineer the code. This technique is called *Fibonacci indent coding*.

Instead of indenting code blocks by a consistent 2/4 spaces, Fibonacci indent coding indents each line according to the Fibonacci series. So, the zeroth lines is indented by 0 spaces, the first line by 1 space, the second line by 1 space, the third line by 2 spaces, the fourth by 3 spaces, the fifth by 5 spaces, and you get the point.



```
int main() {
  int first, second, tmp;

  scanf("%d", &first);
  scanf("%d", &second);

  tmp = first;
  first = second;
  second = tmp;

  printf("First Number = %d\n", first);
  printf("Second Number = %d\n", second);

  return 0;
}
```

```
int main() {
 int first, second, tmp;

 scanf("%d", &first);
  scanf("%d", &second);

   tmp = first;
     first = second;
        second = tmp;

             printf("First Number = %d\n", first);
                     printf("Second Number = %d\n", second);

                                  return 0;
                                                       }
```

Standard indenting (top) and Fibonacci indenting (bottom)

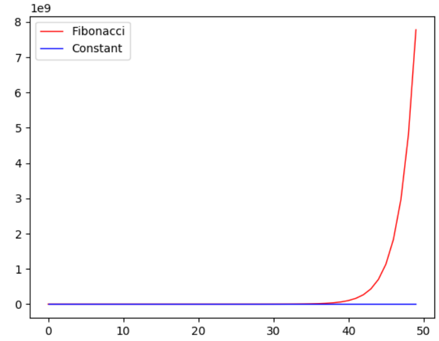
The advantages of Fibonacci indent coding are clear:

1. The increasingly indented lines make it much harder for any third-party to understand and copy the source code, improving *code obfuscation*, as desired.
2. The eleventh Fibonacci number is 89, which is greater than 80, the maximum line width on most editors. Therefore, your code will be scroll past the text editor window starting at the eleventh line, which will confuse most naïve adversaries.
3. Since the Fibonacci sequence grows exponentially, stripping the Fibonacci indenting cannot be done in polynomial time. In other words, even an automated solution to remove the indent coding will take a really long time to do so.
4. Any code snippet that has been copied from StackOverflow or someone else's program will not be Fibonacci indented and obviously distinguishable from the rest of the code. This heuristic can be used to detect software similarity without the use of fancy tools like MOSS, fern, or lichen.
5. It produces a nice staircase pattern in your code.

COMPARISON TO OTHER SEQUENCES

We tested the performance of the Fibonacci indent against 5 other sequences to see which one provides the best performance for our indent coding.

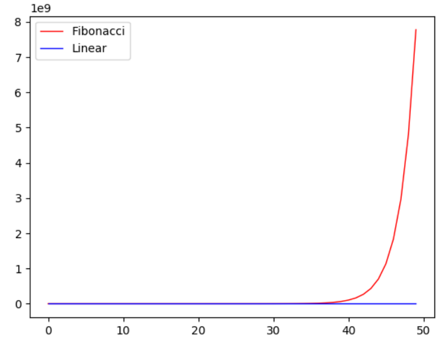
The Constant Series:



```
[ 2 for i in range(50) ]
```

Just using the constant series was equivalent to the naïve 2-space indent coding guidelines we were trying to avoid in the first place.

The Linear Series:

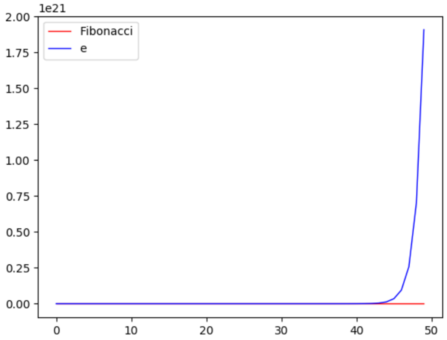


```
[ i for i in range(50) ]
```

The linear series was slightly better, till we were reminded that the Ramanujan summation assigns the sum -1/12 to this series, meaning as we have more and more lines in our code, our file size will converge to a negative number, which is undesirable.

$$1 + 2 + 3 + 4 + \cdots = -\frac{1}{12},$$

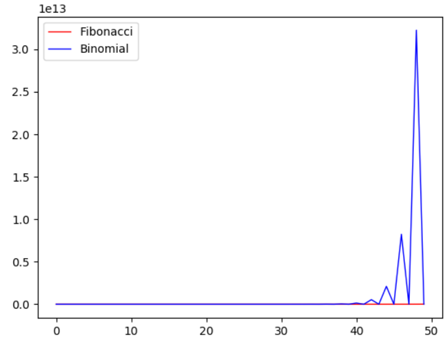
The Exponential Series:



```
[ exp(i) for i in range(50) ]
```

Using e^i gave us the much desired exponential growth, even more so than the Fibonacci series. However, many (all) elements in the series were floating point numbers and it wasn't possible to indent lines by a non-natural number of spaces using the technology available to us today.

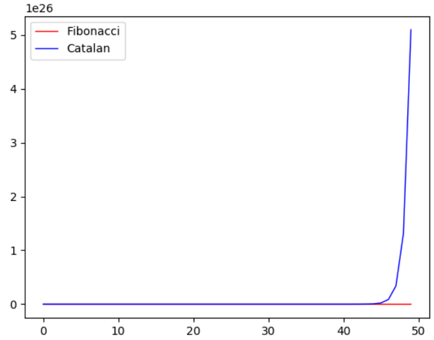
The Binomial Series:



```
[ binom(i, i/2) for i in range(50) ]
```

The binomial series also gave us exponential growth. However, this didn't work for odd numbers because the second term in the binomial coefficient was a decimal (`xx.5`) and gave a `ValueError`, as indicated by the spikes in the graph.

The Catalan Series:



```
[ binom(2*i, i)/(i+1) for i in range(50) ]
```

Using the Catalan numbers for our series avoided the problems we had with the exponential and binomial series. However, we obviously had to reject it because the series name wasn't Italian enough.

PRACTICAL REALIZABILITY

The amazing thing about Fibonacci indent coding is that even though it is an extremely complicated, it is extremely practical. In fact, there already exists an open source Atom package that Fibonacci indents a piece of code.

The package has its limitations though. A couple of them are:

1. It currently supports indenting up to 20 lines, because the Fibonacci numbers grow really large really fast.
2. Python support is still in beta.

CONCLUSION

Overall, we have seen how Fibonacci indent coding can reduce code style and readability and improve code obfuscation. This significantly reduces the risk of academic integrity violations and copyright infringement of your code.

ERROR-DETECTING RLIRFO DATA STRUCTURES FOR THE WIN

Darío de la Fuente García Félix Áxel Gimeno Gil Juan Carlos Morales Vega

March 10, 2019

ABSTRACT

Improvements in the speed and capacity of DRAM have made memory corruption by cosmic rays more likely. In order to ameliorate this issue, we present a recursive stack (Recursive Last In Recursive First Out) that can verify its state by storing all its previous states and the previous states of the previous states up to a desired level of recursion.

Keywords Error detection · Data structures · Cosmic rays · Stacks · Stacks of stacks · Stacks of stacks of stacks · Template Metaprogramming

1 Introduction

13.8 billion years ago, an event that was key for the existence of Sigbovik [1] and your reality [2] itself happened. Yes, we are talking here about the Big Bang, the creation of our universe [3]. Due to several reasons (such as your own existence), the creation of the universe cannot be considered a bad move, despite being different opinions in that topic [4]. However, the universe that was created is nowadays a source of errors in our electronic devices.

In the past, errors due to cosmic rays did not happen since the size of the chips and the transistors were much bigger than the size of a particle, hence a collision was not able to change the state of a bit in memory. However, with the miniaturization of those components, the energy of a cosmic ray has become able to change the state of a bit, causing these types of soft errors to appear [5][6].

To be a bit more precise, the effect of a cosmic ray at the Earth surface is not due to the cosmic ray itself, but rather due to the effect of the subproducts of the ray after interacting with the atmosphere, mainly protons, neutrons and nuclei. Neutron showers have proven to be the main cause of errors due to cosmic rays [7][8].

2 Motivation

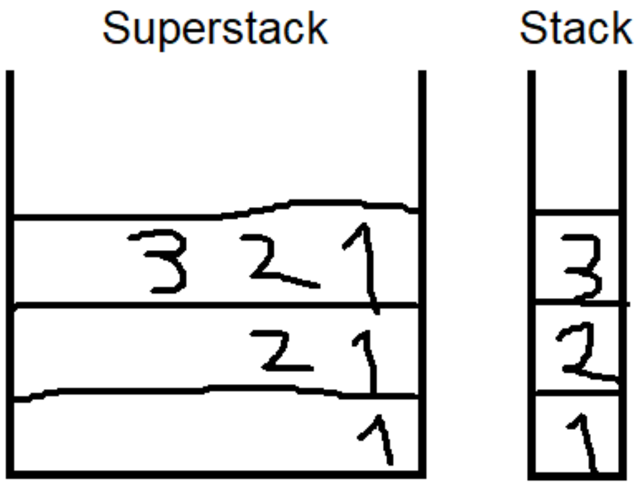
As shown in [9][10][11], the probability of getting an uncorrectable error is quite low, but that is for a small system and the calculations were made back in 2010, when we had 32-45 nm architectures. Nowadays, having 7 nm architectures in some devices, the problem is considerably worse. Of course, the problem is still going to become even worse in the future with more advanced architectures.

Since we cannot go back in time and prevent the universe from being created (as of now), we need an alternative.

Stacks [12][13] (not to be confused with stacks [14] and stack [15]) are one of the most important [16] data structures [17].

3 Idea

Now that we have described the problem, let us explain how to solve it using recursive stacks. The basic idea behind the method is to store all previous states of the stack to be able to search for errors. These states are stored in another stack that we can call the "superstack". The following superb drawing helps for understanding the structure:



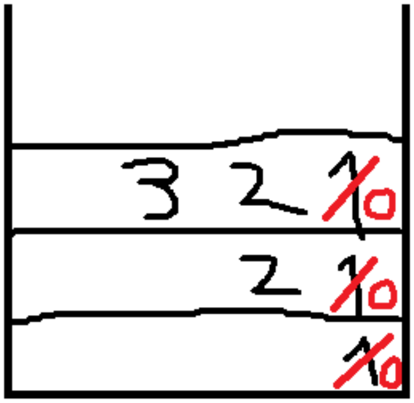
Now, it is easy to check for errors in that structure:

In case there has been a push operation, the first stack of the superstack performs a pop operation and it is compared with the second stack of the superstack. If they are the same, we assume that no error has happened.

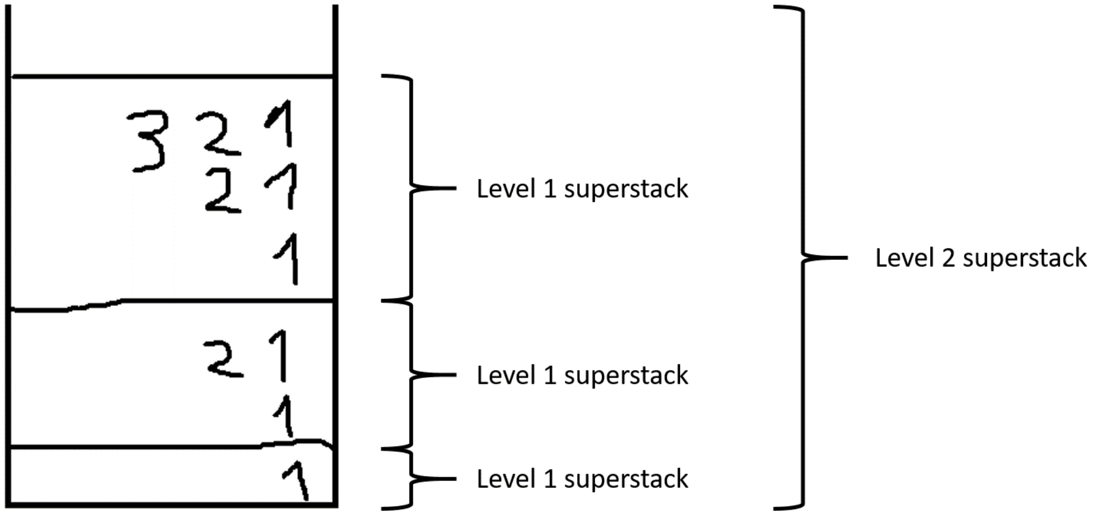
In case of a pop operation, the previous value is added to the top stack in the superstack and then compared with the second one.

As we can see, the push check is straightforward, but the pop check presents a small issue: the top stack does not know which is the popped value (and we cannot just copy the top value of the second top stack). To solve this, we propose a structure pair<stack<T>, optional<T>> for the entries of the superstack, instead of just stack<T>. In the case of a push operation, the optional will take the nullopt value, while in the case of a pop operation, it will take the value of the popped value. This structure also helps to detect if the performed operation was a push or pop very straightforwardly. Moreover, the decision of using an optional here can be considered as an act of good will to promote the features of C++ 17. Cool!

This process is repeated through the full stack until every stack is compared with the previous one. So far so good, but, can we be completely sure that there have been no errors? Not really. What if several evil neutrons decide to team up and corrupt the memory in the specific, following way?

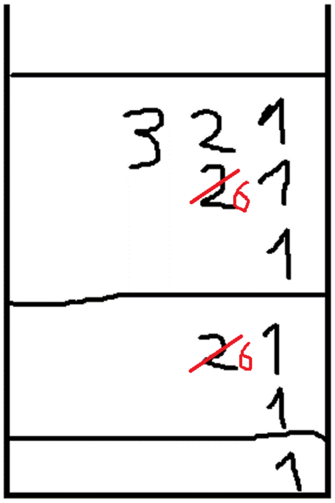


Now we do not have any way of detecting that error. Except that we do have. We can consider the previous superstack as an entry of a bigger superstack. In this way, if an army of evil neutrons decides to perform an all-out-attack over a level 1 superstack, the errors can still be detected in the new level 2 superstack. The structure looks like this:



Checking the errors of a superstack of superstacks is even easier than the previous case. Since a superstack already contains all previous states, there is no need to keep an optional value and the cases for push and pop works exactly in the same way: a pop operation is performed over the top level 1 superstack of the level 2 superstack and the result is compared with the second level 1 superstack.

However, we still need to perform the level 1 superstack check over the top level 1 superstack since there can be errors that cannot be detected at level 2 but they can at level 1:

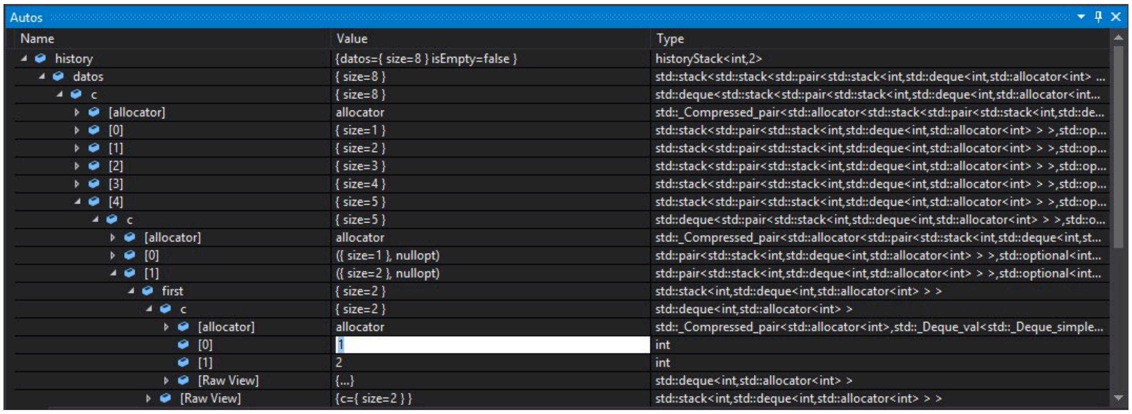


This is starting to look strong, but what if we want to keep some information for longer than what it takes a solar mass black hole to fully evaporate via Hawking radiation? [20] Yes, you know where this is going. In general, we define a level n superstack as a stack of level n-1 superstacks. Detecting an error in a level n superstack is done as explained before: first check the level n superstack and then call the level n-1 checker over the top level n-1 superstack recursively. The inception level n can be freely chosen and the structure is generated using a bit of ~~magic~~ template metaprogramming, as you can see in the code [22].

One could argue that the more inception, the more it will increase the chances of an error happening, but since statistics is a lie, nobody can really prove it.

4 Methodology

We have modeled the cosmic rays using a model that we call "Hand of God". This advanced method consists on using the debug functionalities of the compiler of your choice (Visual Studio 2017 in our case). We set a breakpoint and change random values by hand.

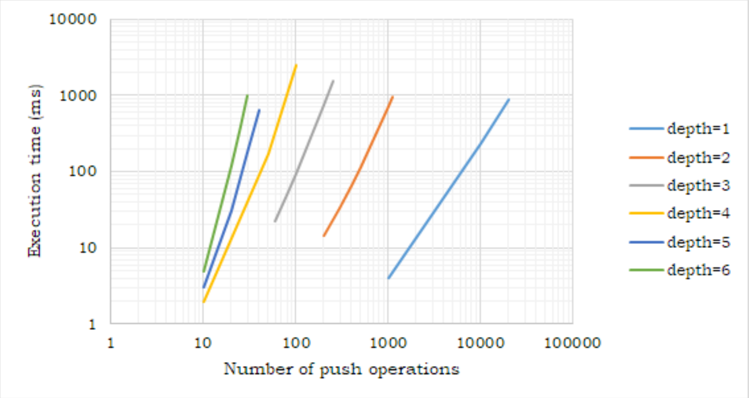


Name	Value	Type
history	{datos={ size=8 } isEmpty=false }	historyStack<int,2>
datos	{ size=8 }	std::stack<std::stack<std::pair<std::stack<int,std::deque<int,std::allocator<int> ...
c	{ size=8 }	std::deque<std::stack<std::pair<std::stack<int,std::deque<int,std::allocator<int...
[allocator]	allocator	std::_Compressed_pair<std::allocator<std::stack<std::pair<std::stack<int,std::de...
[0]	{ size=1 }	std::stack<std::pair<std::stack<int,std::deque<int,std::allocator<int> > >,std::op...
[1]	{ size=2 }	std::stack<std::pair<std::stack<int,std::deque<int,std::allocator<int> > >,std::op...
[2]	{ size=3 }	std::stack<std::pair<std::stack<int,std::deque<int,std::allocator<int> > >,std::op...
[3]	{ size=4 }	std::stack<std::pair<std::stack<int,std::deque<int,std::allocator<int> > >,std::op...
[4]	{ size=5 }	std::stack<std::pair<std::stack<int,std::deque<int,std::allocator<int> > >,std::op...
c	{ size=5 }	std::deque<std::pair<std::stack<int,std::deque<int,std::allocator<int> > >,std::o...
[allocator]	allocator	std::_Compressed_pair<std::allocator<std::pair<std::stack<int,std::deque<int,st...
[0]	({ size=1 }, nullopt)	std::pair<std::stack<int,std::deque<int,std::allocator<int> > >,std::optional<int...
[1]	({ size=2 }, nullopt)	std::pair<std::stack<int,std::deque<int,std::allocator<int> > >,std::optional<int...
first	{ size=2 }	std::stack<int,std::deque<int,std::allocator<int> > >
c	{ size=2 }	std::deque<int,std::allocator<int> >
[allocator]	allocator	std::_Compressed_pair<std::allocator<int>,std::_Deque_val<std::_Deque_simple...
[0]	1	int
[1]	2	int
[Raw View]	{...}	std::deque<int,std::allocator<int> >
[Raw View]	{c={ size=2 } }	std::stack<int,std::deque<int,std::allocator<int> > >

It worked in every case.

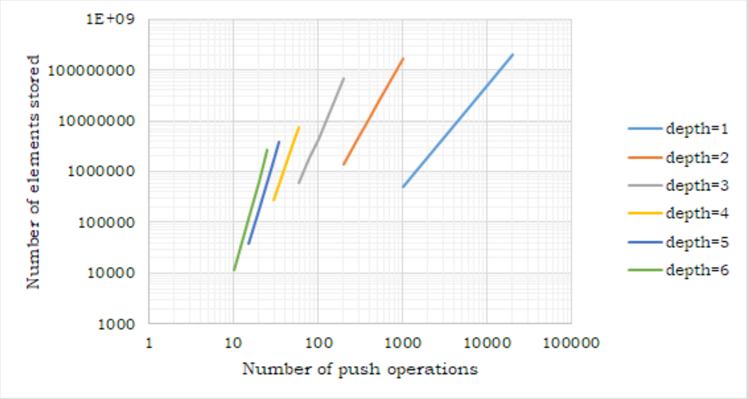
5 Benchmarks

We prepared a benchmark that initialised a stack of depth k and performed on it n push operations. The results can be found in the next graph:



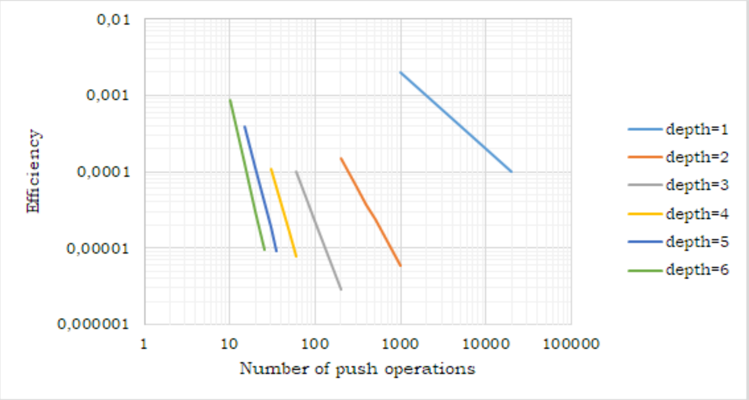
The initial idea was to make longer benchmarks, unfortunately when attempting to do so the virtual machine in which the code was running ran out of memory, thrashing started to happen and the virtual machine had to be force shut down. For that reason, none of the benchmarks go over 3 seconds. Since this is a log-log graph, by looking at the slope of the lines we can deduce easily that the time complexity of performing n operations on a recursive stack of depth k is $\theta(n^{k+1})$. Since this is polynomial time, we can conclude that ~~a mathematician in the wrong field~~ computer scientists will think this is efficient.

In order to better understand this phenomenon, we computed the actual number of elements these benchmarks were producing:



As we can see, at least for levels of recursion of 1, 2 and 3, it is no surprise that the code will explode a virtual machine with 4 GB of RAM assigned to it. As for levels 4, 5 and 6, we suspect that even though the number of elements stored is smaller, there are a lot of smaller level stacks that are being memory aligned and wasting a lot of RAM this way.

By dividing the number of operations by the number of elements generated, we can compute the efficiency:



This result may look absolutely terrible for the untrained eye. However, one must remember that this is absolutely necessary to protect against the evil army of cosmic rays that are out to get you and the correctness of your programs.

Special thanks to all the virtual machines that had to be force shut down. Your sacrifice shall not be forgotten.

6 Potential moral implications for saving erased data

Nobody cares.

7 Future work

Obvious future work includes implementing RLIRFO in languages such as Brainfuck, Unsafe Rust, Safe Rust, Piet and Malbolge. Another imp ovement would be torfork an open source C compiler to use RLIRFO data structure instead of LIFO in call stacks.

Also, in this paper we have talked about data structures that contain its past state. With a similar implementation (probably) we could create data structures that contain themselves, and with that we could create a paradoxical data structure [21] and crash the universe.

8 Conclusion

We have discussed a chaotic-good algorithm to detect errors in a stack that are caused by cosmic rays or other highly probable causes. The data structure that we have discussed can be checked in github. The error detecting algorithm was tested using the “Hand of God” method, yielding the expected results. We have also evaluated the performance of the algorithm and concluded that computer scientists won’t find anything wrong with it. We expect this amazing data structure to be used in the future in almost every device to avoid memory errors in a chaotic way.

Neutrons will not win this crusade.

References

[1] http://sigbovik.org

[2] Your Reality https://www.youtube.com/watch?v=CAL4WMpBNs0

[3] Big Bang https://en.wikipedia.org/wiki/Big_Bang

[4] https://en.wikiquote.org/wiki/The_Hitchhiker’s_Guide_to_the_Galaxy#Chapter_1

[5] Attack of the Cosmic Rays! https://blogs.oracle.com/linux/attack-of-the-cosmic-rays-v2

[6] Serious Computer Glitches Can Be Caused By Cosmic Rays https://science.slashdot.org/story/17/02/19/2330251/serious-computer-glitches-can-be-caused-by-cosmic-rays

[7] Soft Errors https://en.wikipedia.org/wiki/Soft_error

[8] Single Event Upset https://en.wikipedia.org/wiki/Single_event_upset

[9] Do gamma rays from the sun really flip bits every once in a while? https://stackoverflow.com/a/4109288

[10] http://lambda-diode.com/opinion/ecc-memory

[11] T.J. O’Gorman The effect of cosmic rays on the soft error rate of a DRAM at ground level https://doi.org/10.1109/16.278509

[12] Stack https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

[13] Stack https://en.wikipedia.org/wiki/Stack_(C%2B%2B)

[14] Stack https://en.wikipedia.org/wiki/Stack_(mathematics)

[15] Learn to Stack https://youtu.be/Xqpf1FxU2q4?t=291

[16] What Is a Full Stack Developer https://www.youtube.com/watch?v=UtDpYVf9jKU

[17] List of data structures https://en.wikipedia.org/wiki/List_of_data_structures

[18] Double Byte Error Detecting Codes for Memory Systems https://doi.ieeecomputersociety.org/10.1109/TC.1982.1676056

[19] Population dynamics https://wiki.puella-magi.net/Population_dynamics

[20] https://www.quora.com/How-long-would-it-take-for-an-Earth-mass-black-hole-to-evaporate-due-to-Hawking-radiation

[21] https://en.wikipedia.org/wiki/Russell%27s_paradox

[22] RLIRFO https://github.com/juancarlosmv/RLIRFO

Measurement studies

34 Applications of Standard ML at Google

Andrew Benson

Keywords: SML, Standard ML, one-fifty, Harper, Google, software engineering, programming languages

35 93% of paint splatters are valid Perl programs

Colin McMillen and Tim Toady

Keywords: perl, OCR, regret

36 A survey and projection of SIGBOVIK trends

Jenny H. Lin

Keywords: SIGBOVIK, self referential, “Figures”, join the committee

Applications of Standard ML at Google

Andrew Benson
Google
adbenson@google.com

Recitation Section SWE
Out: Sunday, March 3, 2019
Due: Monday, April 1, 2019 at 17:30 EST (?)

1: Abstract (15%)

We present a collection of Google projects using Standard ML, as well as lessons learned while introducing SML engineering at Google.

Keywords: SML, Standard ML, 150, Harper, Google, software engineering, programming languages

2: Introduction (50%)

Standard ML (commonly known as SML) is a general-purpose functional programming language used primarily to instruct freshman computer science majors at specific Pittsburgh-based universities. Among its key strengths are strong compile-time type-checking, Turing completeness, and ability to compile to JavaScript[1]. The simplicity and beauty of algorithms written in SML has won it a considerable fanbase concentrated along Forbes Avenue[2], and it doesn't hurt that its syntax is easy to write *operator and operand don't agree [tycon mismatch]*.

Google is an American software company headquartered in a really large number[3]. Product domains of Google include search engines, self-driving engines, mobile operating systems, stationary operating systems, big data, small data, and more. Since Google's software engineering is widely considered to be of high caliber, it is clear that CS students at Carnegie Mellon could learn a lot about how to write production-ready and scalable SML from a study of existing uses of it within Google.

We present an overview of each and every product at Google utilizing SML, the benefits we reaped during the transition to SML, and advice from seasoned SML engineers at Google.

3: Each And Every Application Of Standard ML At Google, All Benefits Reaped During All Transitions We Made To Standard ML, And Advice From All Of Our Seasoned Standard ML Engineers at Google (0%)

4: Future Work (20%)

We know for a fact that for every application, benefit, or piece of advice given in the last section, the reader gained a lot of knowledge. As a result, we are considering studying the use within Google source code of another concept taught to freshman computer science majors at specific Pittsburgh-based universities: dynamically checked *contracts*, such as preconditions and postconditions. The author has personally written contracts[4] in production Java code running on Google Search, so we expect the results of such as study to be enlightening.

5: Notes (8%)

The author would like to note that SML is an ambiguous acronym, and Google is very much interested in Standard Machine Learning, Supervised Machine Learning, etc.

The author does like SML andalso prefers OCaml.

6: References (7%)

[1]: https://www.smlserver.org/smltojs/

[2]: Coincidentally, Carnegie Mellon University's canonical address is 5000 Forbes Ave.

[3]: $10^{10^{100}}$

[4]: https://github.com/google/guava/wiki/PreconditionsExplained

CONFIDENTIAL COMMITTEE MATERIALS

SIGBOVIK 2019 Paper Review

Paper 19: Applications of Standard ML at Google

Stefan Muller
Former 15-150 Instructor
Expensive Building, Forbes Avenue
Rating: 42.0
Confidence: `val it = 42.0 :  real`

This paper gives an excellent summary of the many benefits of Standard ML, a strongly-typed, usable, production-quality language, as well as a number[1] of applications of it in practice in Google, a reasonably extensive search engine codebase developed by researchers at Stanford.

I agree with the author that the publication of this study could be a great learning experience to CS students at Carnegie Mellon, as well as a number of other universities that make extensive use of Standard ML[2].

As future work, the author may wish to consider even further expanding the use of Standard ML at Google. Its strong abstraction properties, for example, could be useful to help Google avoid the privacy issues that have plagued other tech companies in recent days. Consider the code on the following page.

[1] zero is a number

[2] zero is a number

```
signature User =
sig
    val name           : string
    val age            : int
    val interests      : string list
    val docs           : string list
    val sheets         : string Array.array Array.array list
    val search_history : string list
    val emails         : string list
    val contacts       : (string * int) list
end

signature AdUser =
sig
    (* All the info I want Google to give to advertisers *)
end

structure Me :> AdUser =
struct
...
end


functor Ads (U : AdUser) =
struct
...
end
```

Conflict of interest disclosure: The reviewer, while generally distrustful of Big Tech, uses Google products when necessary (read: always).

93% of Paint Splatters are Valid Perl Programs

Colin McMillen and Tim Toady
twitter.com/mcmillen & famicol.in/sigbovik

The authors assure us the materials will be available “soon” from: `http://famicol.in/sigbovik`

Abstract

In this paper, we aim to answer a long-standing[1] open problem in the programming languages community: *is it possible to smear paint on the wall without creating valid Perl?*

We answer this question in the affirmative: it **is** possible to smear paint on the wall without creating a valid Perl program. We employ an empirical approach which finds that merely 93% of paint splatters parse as valid Perl. We analyze the properties of paint-splatter Perl programs, and present seven examples of paint splatters which are *not* valid Perl programs.

Background

In a February 2019 Twitter conversation, Adrienne Porter Felt expressed a desire for her kid to smear paint on the wall instead of learning vocational skills such as computer programming [1]. In response, Jake Archibald posed the question which forms the basis of this research: “is it possible to smear paint on the wall without creating valid Perl?” [2]

While many PL researchers have (often derogatory) folk beliefs about the Perl programming language, the language itself has not been the subject of much formal academic inquiry. One exception is Jeffrey Kegler’s proof that the Perl programming language is undecidable [3], often summarized with the maxim “only `perl` can parse Perl”.

Another maxim of the Perl community is “There Is More Than One Way to Do It.”[2] Despite the popularity of this maxim in the Perl community, the authors are not aware of any professional Perl engineers whose development practices involve smearing paint on walls.

We thus believe that we are the first researchers in academia or industry to directly address the question of whether paint splatters are valid Perl programs.

Experimental Setup

Our approach to answering this question is an empirical one. Given an input image, we run optical character recognition (OCR) software on that image to extract candidate text. As previously mentioned, the question “is this string valid Perl?” is theoretically undecidable; we therefore fed the extracted text into the `perl` executable (version `5.26.1`) to check whether the OCR’d string corresponded to a valid Perl program.

We are not aware of the existence of any standard paint-splatter datasets in the object recognition or OCR communities. Also, ImageNet’s website was down on

[1] By “long-standing”, we mean “for roughly a month or so”.

[2] Often pronounced “Tim Toady”; hence the name of the fictional second author of this paper. (Like many researchers, we collaborate with other authors primarily so that our use of the royal “we” doesn’t come across as pretentious.)

The authors assure us the materials will be available "soon" from: http://famicol.in/sigbovik

the day that we decided to perform this research. We therefore paid an unemployed person[3] to download 100 examples of paint-splatter artwork by searching Pinterest using the query "paint splatter wallpaper".

We manually filtered out all images with any form of overlaid or watermarked synthetic text, because the Perl program (?) "iStock by Getty Images" is not particularly interesting.

The resulting 100 images are shown in Figure 1, and are also available online as supplementary material to this paper (see the Addendix).

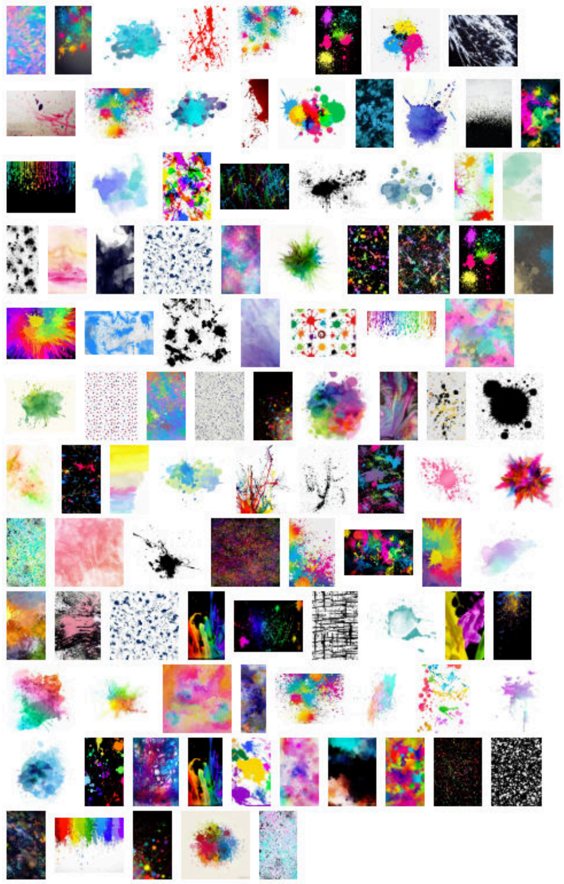


Figure 1. Paint-splatter image dataset.

To perform OCR on the input ima used the Tesseract OCR engine (v 4.0.0-beta.1) [4]. Tesseract is an open-source OCR library that provides two separate algorithms for optical character recognition: a "legacy" engine that uses traditional OCR techniques, and a newer engine based on LSTM models. Tesseract also provides a third OCR engine which somehow combines the two. Unfortunately, the documentation does not describe this mode in detail, but we chose to make use of it anyways.

Additionally, Tesseract provides multiple algorithms for performing page segmentation. For example, page segmentation mode #4 assumes that the page consists of a single (possibly multiline) column of text; mode #7 treats the image as a single line of text.

It is possible to configure Tesseract's legacy OCR engine with a list of allowed characters expected to be in the input alphabet. Since Perl programs mostly consist of the printable subset of ASCII, we restricted the OCR engine alphabet to ASCII characters in the range [32, 127). Tesseract's newer LSTM-based engine doesn't appear support this sort of configuration.[4] We also disabled features related to language modeling where possible (for example, penalties for "words" that are not found in the English dictionary).

It was unclear to the authors which OCR engine and page segmentation modes would best correspond to "splats of paint

[3] The first author.

[4] Apparently in the brave new world of neural nets, Anything Goes™ (except for the ability to configure things.)

that might parse as valid Perl programs”. Therefore, for each image, we tried all combinations of the 3 OCR engines and 4 of the 14 different page segmentation modes (modes #4, #7, #8, and #9) to determine which configuration was the most fruitful for producing a valid Perl program out of that specific image.

This “try multiple algorithms until one of them happens to work” approach is profoundly unethical — especially since we don’t have separate training, test, and validation sets — but at least we’re being honest about what we’re doing, instead of inventing a fancy-but-obfuscatory technical term like “ensemble methods” or “hyperparameter tuning”.

Results

The main result of this paper is that 93 of 100 images in our dataset successfully parsed as valid Perl programs under at least one combination of Tesseract OCR engine & page segmentation mode. (It is worth explicitly noting that we only considered non-empty Perl programs as successes.)

The most fruitful single combination of parameters is provided by the LSTM engine using page segmentation mode 9, which successfully produces valid Perl programs out of 55% of paint splatters.

The pure LSTM approach was the most successful of the three OCR engines, with 74% of input images successfully parsing as Perl under at least one of the four page segmentation modes. The legacy OCR engine succeeded in finding valid Perl programs on 62% of images, while the combination legacy+LSTM succeeded merely 40% of the tim unclear why the combination of the two OCR algorithms would be significantly worse at recognizing valid Perl programs than each OCR algorithm on its own.

Discussion & Analysis

While we successfully found valid Perl programs in 93 of 100 input images, all of these programs are uninteresting; they don’t actually seem to *do* anything when executed. However, some of them do evaluate to a value, which is then discarded without being displayed. [5]

We therefore passed each valid program into Perl’s `eval()` function and printed out the result of evaluating it. Many of these are simple integer literals. Figure 2 shows an input image which is read by OCR as the string `“35”`, which evaluates to the number `35` when parsed by Perl.

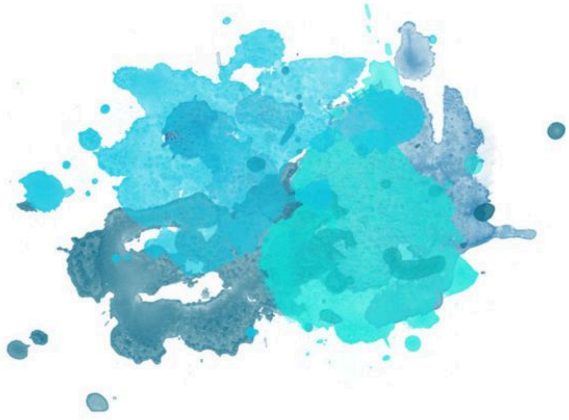


Figure 2. If you squint, you can see the number `35`.

Figure -3 shows a somewhat more interesting case. This input image is read by OCR as the string "- 3", which evaluates to the number -3 when parsed by Perl.

The authors assure us the materials will be availab

Figure -3. -3.

In all, the dataset contains 20 images which can be parsed as numeric literals. An interesting thing about these images is that they're not particularly Perl-specific; for example, the program shown in Figure -3 also evaluates as the number -3 in Python (and presumably several other programming languages.)

-3 is the smallest number resulting from the output of our valid Perl programs; the program which evaluates to the largest number is shown in Figure 4.

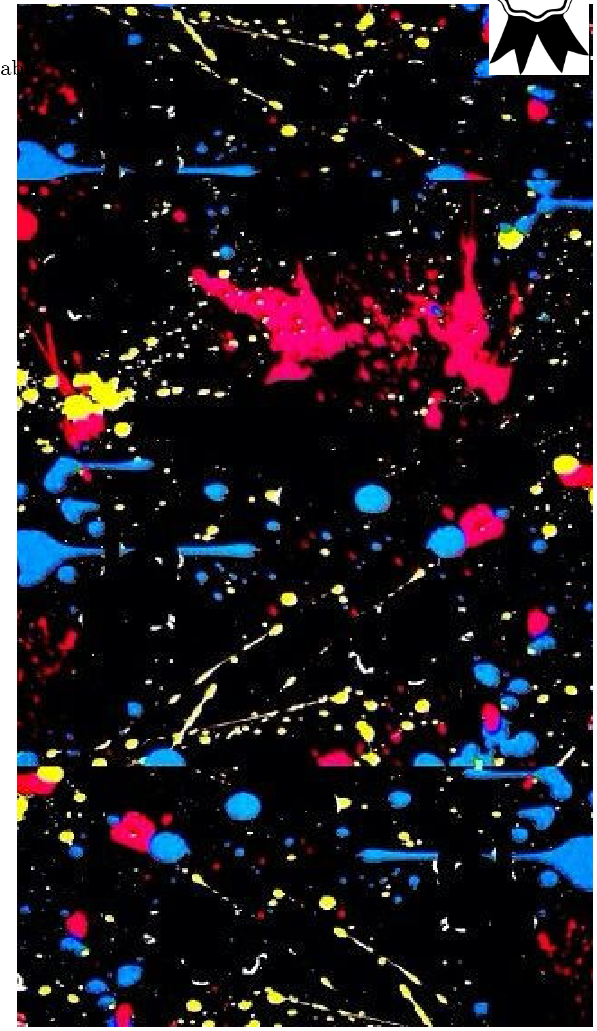


Figure 4. The valid Perl program "225252", which evaluates to 225252.

The remainder of the valid Perl programs are mostly valid due to a Perl language feature that is *not* commonly present in most programming languages. Namely, Perl has a feature called "unquoted strings", in which a sequence of alphanumeric characters by itself is parsed as though it were a quoted string. As an example, Figure 5 is read by OCR as the text ME, which evaluates to the string "ME" even though the ME isn't quoted. This would result in a syntax error in most other programming languages.

The authors assure us the materials will be available "soon" from http://famicol.in/sigbovik

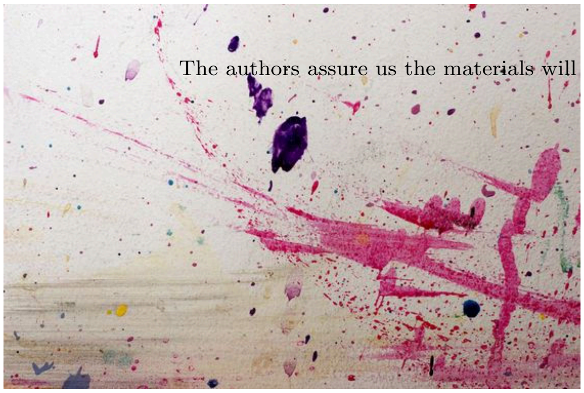


Figure 5. It ME.

Figure 6 represents the string "gggijgziifiiffif", which by pure coincidence happens to accurately represent the authors' verbal reaction upon learning that "unquoted strings" were a feature intentionally included in the Perl language.[5]

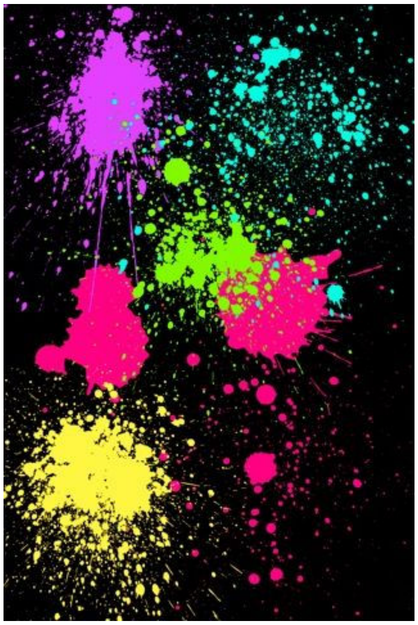


Figure 6. gggijgziifiiffif!

(To be fair to Perl, when perl is ru the -w flag to enable warnings, helpfully inform the user that at some point in the future, the Perl developers will most likely pick gggijgziifiiffif as a new reserved word:

```
Unquoted string
"gggijgziifiiffif" may clash
with future reserved word at -
line 1.)
```

Another interesting case is presented in Figure 7. This image represents the source code ";" which is non-empty, but which is just a statement separator that does not evaluate to anything.

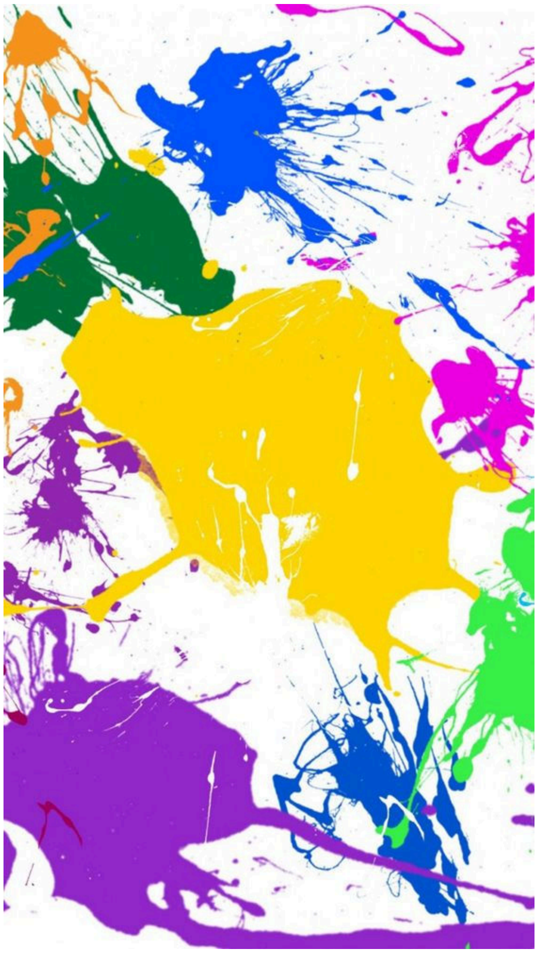


Figure 7. A statement of no purpose.

[5] This feature *does* enable a neat quine: the Perl program "Illegal division by zero at /tmp/quine.pl line 1.", when saved in the appropriate location, outputs "Illegal division by zero at /tmp/quine.pl line 1." The reason for this behavior is left as an exercise for the reader.

The Rogue’s Gallery

The authors assure us the materials will be available “soon” from: http://famicol.in/sigbovik

At this point, we would not blame the reader for sympathizing with Jake Archibald’s conjecture that *all* paint splatters are in fact valid Perl programs. Perhaps Adrienne’s kid is doomed to accidentally write valid Perl even when just trying to smear some paint around at random. Here we finally present some counterexamples: the seven images in our dataset which do not, under any OCR interpretation, parse as valid Perl.

Figure 8 presents a splatter which is read by OCR as any of the following strings:

- `fifi;%:'i1i:`
- `.%f:`
- `&`
- `i;%:’;;:`

Surprisingly, none of these strings represent valid Perl programs.

Figure 8. Not valid Perl. Obviously.

For completeness, we present the six such images as Figure 9.

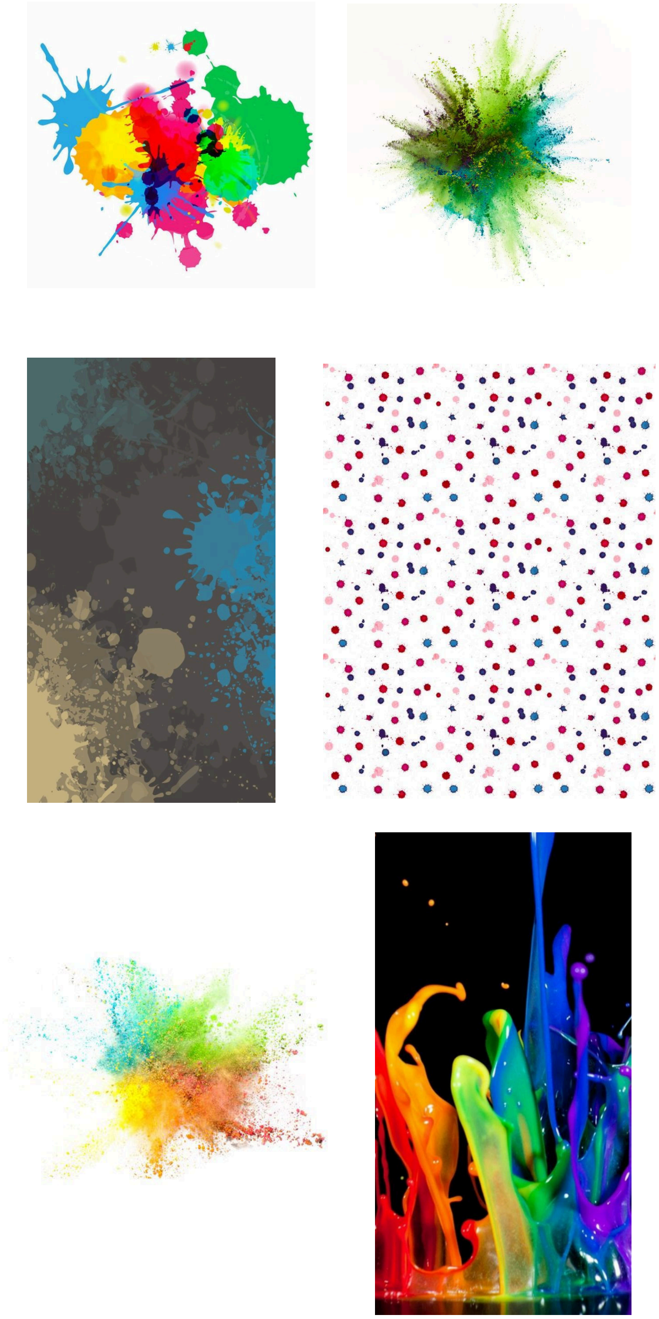


Figure 9. Finally, we are free of the tyranny of accidentally writing valid Perl programs. These are the sorts of paint splatters one might want one’s child to produce when they’re just having fun.

Woomy?!?

The authors assure us the materials will be available "soon" from: http://famicol.in/sigbovik

Fans of the *Splatoon* video game series will naturally be wondering whether the “splats” in Nintendo’s official game artwork are also valid Perl programs. Our preliminary answer is *yes*: the image in Figure 10, downloaded from Nintendo’s official Splatoon website [6], successfully parses as a valid Perl program.

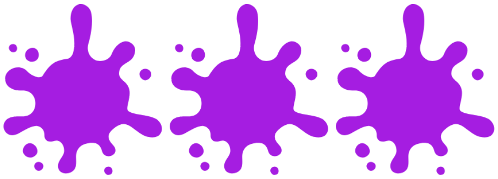


Figure 10. Splat splat splat… woomy?

Given Nintendo’s family-friendly image, the authors were surprised to find out that the source code that results from OCR’ing this image is somewhat NSFW.[6] We thus we elide it here, for the sake of SIGBOVIK’s younger readers.

Future Work

While the results presented in this paper are novel and important, they only begin to break ground on what could be a very fruitful area of further research.

The dataset used in this pape relatively small dataset of on paint-splatter images. It would be good to confirm these results on a larger dataset, and with a greater variety of images. Perhaps next time ImageNet won’t be down.

We also noticed far too late that while the original question referred to paint *smears*, we elected to search Pinterest only for paint *splatters*. It is unclear at whether these results would change significantly for paint splatters vs. paint smears.

Similarly, our choice to select images from Pinterest ensured that they were reasonably high-quality paint splatters, as at least one Pinterest user had chosen to “pin” that image as something worth saving for later. It would be worth investigating whether amateurish, lower-quality paint splatters — such as those produced by a young child — are less likely to be parsed as valid Perl programs.

After downloading the 100 images used in our dataset, Pinterest somehow inferred that the authors of this paper might be interested in images of “swimwear trends”. We have not yet investigated whether 2019’s latest swimwear trends are more or less likely to parse as valid Perl programs.

[6] Really, it’s Perl itself that’s most unsafe for work.

The authors assure us the materials will be available “soon” from: `http://famicol.in/sigbovik`

Addendix

The source code for the research presented in this paper, as well as the full dataset of 100 paint-splatter images & the result of evaluating each, will soon be available at http://famicol.in/sigbovik.

References

[1] Adrienne Porter Felt. https://twitter.com/__apf__/status/1095698777300586496.

[2] Jake Archibald. https://twitter.com/jaffathecake/status/1095706032448393217.

[3] Jeffrey Kegler. “Perl Is Undecidable”. The Perl Review, Volume 5, Issue 0, Fall 2008, pp. 7-11. Available online at http://www.jeffreykegler.com/Home/perl-and-undecidability.

[4] Tesseract Open Source OCR Engine. https://github.com/tesseract-ocr/tesseract

[5] `Patrician|Away` and `bovril`, personal correspondence. Recorded at http://bash.org/?240849.

[6] Splatoon 2 amiibo™ home page. https://splatoon.nintendo.com/amiibo/.

A Survey and Projection of SIGBOVIK Trends

Jenny H. Lin
jennylin@andrew.cmu.edu

ABSTRACT

SIGBOVIK is a prestigious conference with a long history of outstanding research with at least token jokey content in a non-zero number of axes. Yet to take for granted continued future excellence is the height of folly. This paper seeks to document trends of SIGBOVIK past in hopes of guiding future members of the organizing committee as well as any other potential joke conferences. It includes several graphs, tables, and even a survey of several past general chairs. It also attempts to recruit 2020's general chair, since by tradition no one person may chair SIGBOVIK twice. So join the SIGBOVIK organizing committee! That's right you! I'm talking to you! Don't just skip past the rest of the abstract and go look at the figures this is important SIGBOVIK can't continue without your support make sure to email the org-list and— hey!

KEYWORDS

SIGBOVIK, self referential, "Figures", join the committee

ACM Reference Format:
Jenny H. Lin. 2019. A Survey and Projection of SIGBOVIK Trends. In *BOVIK '19: ACM Symposium on Computational Heresy, April 1, 2019, Pittsburgh, PA.* ACM, New York, NY, USA, 4 pages. https://doi.org/notrealdontclick

1 INTRODUCTION

SIGBOVIK is a conference on humorous research that, as of this paper's writing, is celebrating its 13th year of continuous publication. While it does not claim to be the first or the best or the longest running joke conference[1], it does claim to be older than at least one joke conference [1]. This paper thus serves as one part statistical analysis, one style guide for future joke conferences, BOVIK or otherwise, with a dash of recruitment for future SIGBOVIX. Contact sigbovik@gmail.com today to join the organizing committee. Mention this paper and get ϵ% off your next proceedings purchase today.

2 PRIOR WORK

Previous SIGBOVIK Messages from the Organizing Committee have included analysis of trends such as paper distribution across the four quadrants [2], and acceptance rate [3]. However, none of these were actual papers, so they don't count. Furthermore this is the first work to include a survey of SIGBOVIK experts; after all, who is more qualified to pontificate about the future of the conference than those who have lead it in the past? Probably the people who execute the actual drudgery of executing the conference, but there is some overlap so we're probably still fine [figure 1].

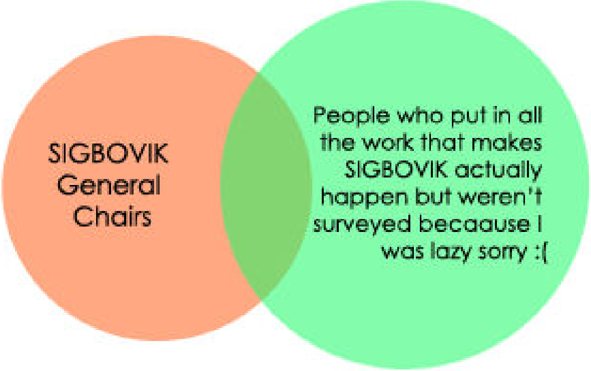

SIGBOVIK General Chairs
People who put in all the work that makes SIGBOVIK actually happen but weren't surveyed becaause I was lazy sorry :(

Figure 1: A venn diagram of SIGBOVIK general chairs and people who actually contribute to making the conference happen. A non-zero number of people from the intersection were included in the survey

3 PAPERS

The piece de resistance of any conference is its papers, and SIGBOVIK has had a lot of them. 371, to be exact. However, it is well understood that the true measure of a paper's merit is its number of words. How loquacious are SIGBOVIK papers? The answer is quite garrulous indeed, especially if you don't even bother trying to filter out the weirder bits. Most importantly, 2019 has the most words of all [figure 2].

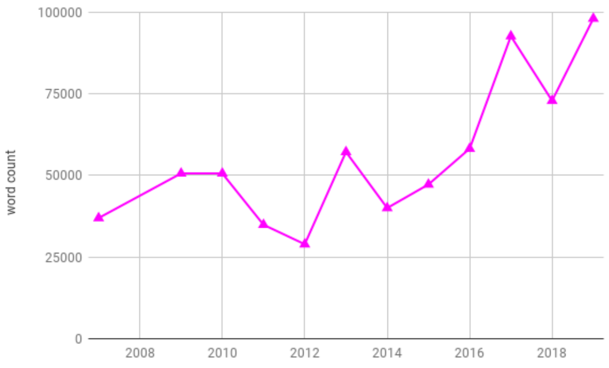


Figure 2: An overall wordcount of SIGBOVIK proceedings over the years, with 2019 as the clear winner.

In addition, figures are known to contain at least a thousand words. However, distinguishing figures from various images, comics, and other incidental graphics that have appeared in SIGBOVIK is difficult. Thus we instead we count the number of times the word "Figure" appears in the proceedings instead. As one might "Figure",

[1] mostly because the author does not feel like verifying or disproving such claims

SIGBOVIK '19, April 1, 2019, Pittsburgh, PA
2019. ACM ISBN 000…$0.00
https://doi.org/notrealdontclick

it appears a lot, and it appears the most in 2019. The 2019 proceedings also have the most pages[2]. In fact, it has almost as many pages as it does "Figure"s. One might say, a "Figure" for every page [figure 3].

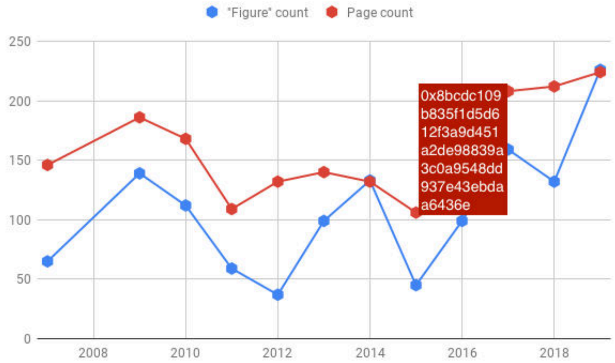


Figure 3: A graph of "Figure" and page counts, with 2019 as the even clearer winner.

Of further interest is seeing the distribution of papers across the four quadrants of research: Humorous treatment of humorous ideas, humorous treatment of serious ideas, serious treatment of humorous ideas, and technically not supposed to be in this conference but we'll take it anyways (henceforth known as "Other"). Fortunately, this process is quite trivial and well documented in previous proceedings [?]. Thus it is left as an exercise for the reader to distribute the papers, perhaps by using a number 2 pencil [table 1].

4 DATES

The word date has many definitions, including: the brown, oblong edible fruit of a palm (Phoenix dactylifera); a social engagement between two persons that often has a romantic character; and the time at which an event occurs [7]. Though SIGBOVIK has papers on both fruit [8] and romance [4–6], this section will focus solely on the third definition, leaving all fruit and/or love surveys for future work.

It has been oft said that the traditional date for the conference is April 1st. Indeed, looking at the distribution of dates, we can see the highest peak on April 1st [figure 4]. That said, the conference has fallen on different days before, mostly out of respect of various spring holidays and/or annoyance with conflicting events. Thus it is up to the organizing committee's discretion to choose the conference date, so long as they continue to keep the conference date in April. SIGBOVIK only ever happens in April.

It is a longstanding tradition for SIGBOVIK to include three deadline dates: an initial deadline, an extension, and a final extension, for all the procrastinators. But how longstanding is longstanding? A look into the first SIGBOVIK webpage (2007) reveals no mention of any deadline extensions. Instead, their list of dates only include an "Author Notification" date (March 15) and Final Abstract due

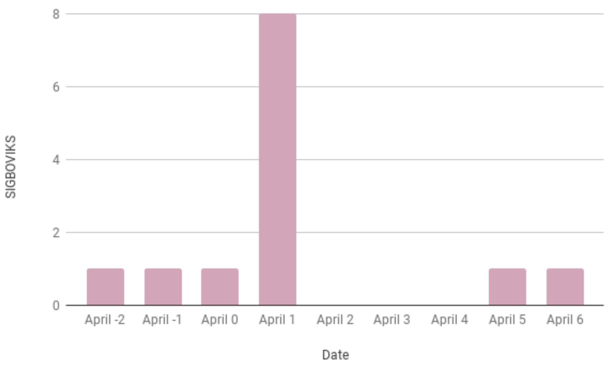


Figure 4: A Bar chart of conference dates $stdev = 2.1$.

date (March 30). This quickly fell out of practice, as SIGBOVIK is so prestigious that all submissions are of top quality and the author notification is almost always "accepted"[3], and needing to put together the proceedings in two days must have sucked. In 2008, we see mention of a "Next SIGBOVIK paper submission deadline" and "Probably the last SIGBOVIK paper submission deadline", implying at least three deadlines occurred in 2008. Unfortunately, there is no mention of any earlier deadlines. From 2009 onwards, the existence of the SIGBOVIK-announce mailing list combined with the websites made it relatively straightforward to acquire the deadlines, though *some* years neglected to mention their initial deadlines even in their initial call for papers [figure 5].

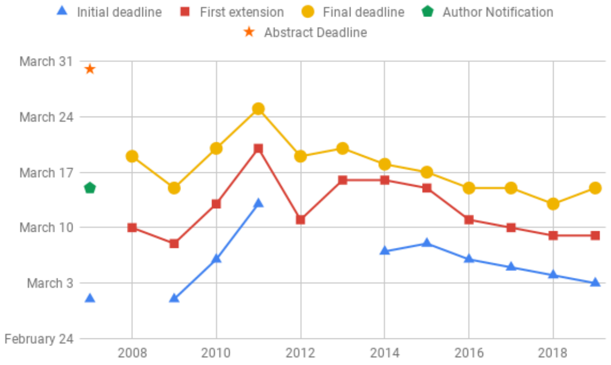


Figure 5: A (dead)line graph of all the initial, extended, and final deadlines. Note the limited edition "author notification" and "abstract deadline" deadlines, found only in 2007.

5 PROCEEDINGS

After the final deadline but before the conference date, papers are scrupulously reviewed and then accepted. Acceptance rates historically have fallen in the range [ϵ, 100], inclusive. Accepted papers are then divided into tracks [figure 6]. Note the multiple

[2] The observant might notice 2008 missing from these plots. Rest assured the Association for Computational Heresy is working tirelessly to rescue the 2008 proceedings from the glitch in space-time, and it would have lost at all these statistics anyways

[3] except when it is rejected, see SIGBOVIK 2017

Table 1: Breakdown of SIGBOVIK papers across quadrants

	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
Humorous Treatment of Humorous Ideas													
Humorous Treatment of Serious Ideas													
Serious Treatment of Humorous Ideas													
Other													

discontinuities (due to a combination of space-time disruptions and non-numerical tracks), which make the track number function nondifferentiable. This is probably why the proceeding's chair is having a harder and harder time binning the papers (see: chess).

The bound copies of the proceedings all have covers which include at least one color. Some covers even have more than one color. If we consider the distribution of cover colors 7, we see that there are large amounts of green (23.1%), periwinkle (20.2%), and faces(14%). Hence a representative cover would contain all three. However, SIGBOVIK is a conference welcoming of all colors, and far be it for us to restrict the cover artist from drawing as they please, mostly when their office is right next to the author's. Nice job with 2019's cover by the way!

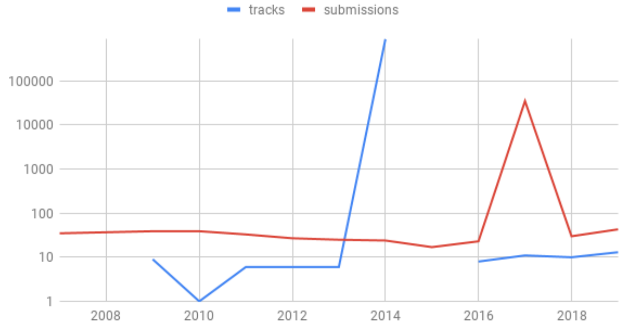


Figure 6: A log scale line graph (not area graph) of number of tracks and number of submissions.

6 WEBSITE

All the SIGBOVIK websites are good, but 2019's is the best. It has the most pages, and even if it didn't it would still objectively be the best because it just is. Citation *not* needed.

7 CHAIRS

Approximately twelve hours before the 2019 final deadline, the author thought it would be fun to write a survey and send it to past, present, and (one) future chair). Of the fourteen contacted chairs, eight responded. All survey questions were optional, and thus not all questions have the same number of responses.

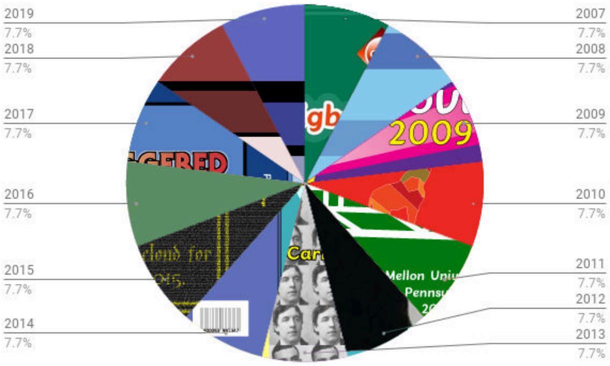


Figure 7: A pie chart of all the SIGBOVIK covers, even ones that no longer exist in this timeline

To get the participants familiar with the survey interface, they were asked a series of questions related to SIGBOVIK (See appendix). They were also quizzed on the name of 2007's general chair. Of note is that all the participants got the answer wrong; the answer is not "Before my time", "$\Theta(Tom7)$", or "me and jason", it is the definitely non-fictional Red Franz, as can be seen on 2007's website.

It is known that all the SIGBOVIK chairs are really cool people, however, the coolness of 2020's chair is yet unknown. Participants were thus asked for the coolness rating of 2020's chair [figure 8]. One out of seven participants gave the 2020 chair a five star coolness rating. The remaining six were split between a three star rating (but of the michelin kind) and a $-35C$ windchill rating. Thus the 2020 chair will likely be highly rated among restaurants while also carrying significant risk of frostbite. That said, 2020's chair will not be "Coolest of them all", a title which presumably belongs to some other chair. Followup work may be required to investigate who is the actual coolest.

Participants were also asked to rate the ergonomicity of all other chairs, including the 2020 chair, on a scale from 0 to 7. However, on further reflection, the author realized publishing this data might cause extreme Drama, as ergonomics is very serious business. Thus the only data published shall be that the 2020 chair has a projected ergonomicity of 4.25.

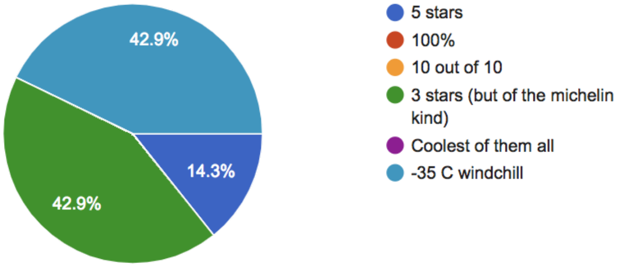


Figure 8: A pie chart of the 2020 chair's projected coolness.

8 CONCLUSION

SIGBOVIK is a really great conference and y'all better help with future proceedings or I'll be upset. 2019 was a great year

ACKNOWLEDGMENTS

Sol I owe you one drawing of a twenty dollar bill.

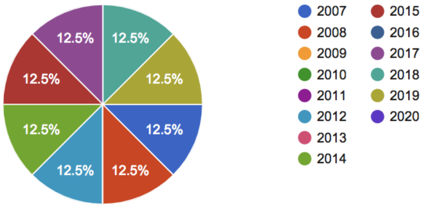
REFERENCES

[1] [n. d.]. AHAHA 2019.
[2] 2014. *SIGBOVIK 2014 Message From the Organizing Committee.*
[3] 2014. *SIGBOVIK 2017 Message From the Organizing Committee.*
[4] Rose Bohrer. 2017. Call For Partners: Romance with rigor. In *Proceedings of the 11th. ACM SIGBOVIK symposium on Principles of Programming Languages.* 28–33.
[5] Rowan Copley. 2017. Towards A Well-Defined and Secure Flirtation Protocol. In *Proceedings of the 11th. ACM SIGBOVIK symposium on Principles of Programming Languages.* 15–21.
[6] Nicolas Feltman. 2013. Optimal Coupling and Gaybies. In *Proceedings of the 7th. ACM SIGBOVIK symposium on Principles of Programming Languages.* 54–58.
[7] Merriam-Webster. 2019. date. https://www.merriam-webster.com/dictionary/date
[8] Aidan Gomez Nick Frosst. 2019. Towards Automatic Low Hanging Fruit Identification For the Steering of ML Research. In *Proceedings of the 13th. ACM SIGBOVIK symposium on Principles of Programming Languages.* 81–84.

SURVEY RESPONSES

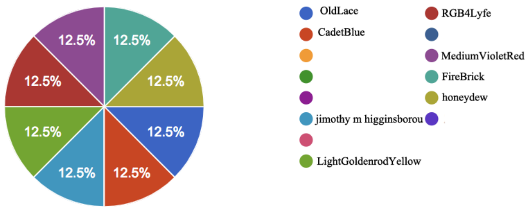
Year Chaired

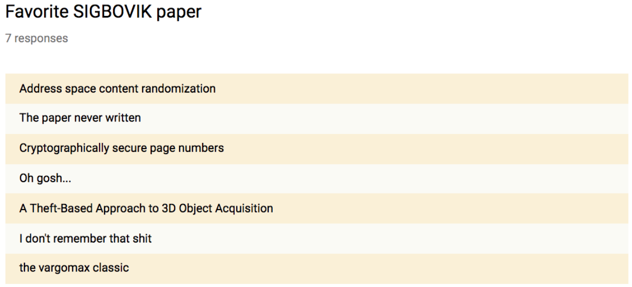
8 responses



Favorite CSS Named Color

8 responses





Favorite SIGBOVIK paper

7 responses

Address space content randomization

The paper never written

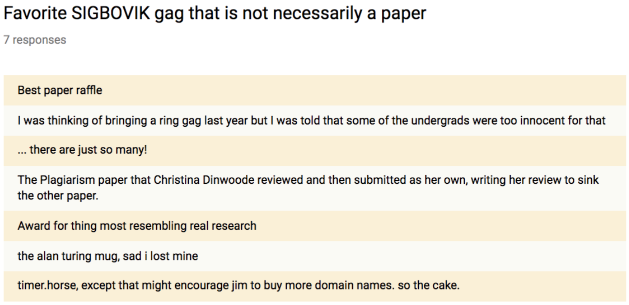
Cryptographically secure page numbers

Oh gosh...

A Theft-Based Approach to 3D Object Acquisition

I don't remember that shit

the vargomax classic



Favorite SIGBOVIK gag that is not necessarily a paper

7 responses

Best paper raffle

I was thinking of bringing a ring gag last year but I was told that some of the undergrads were too innocent for that

... there are just so many!

The Plagiarism paper that Christina Dinwoode reviewed and then submitted as her own, writing her review to sink the other paper.

Award for thing most resembling real research

the alan turing mug, sad i lost mine

timer.horse, except that might encourage jim to buy more domain names. so the cake.

Back to the future

Need more RAM? Just invent time travel!

Robert Andrews*[1], Mitchell Jones*[1], Patrick Lin*[1], and the UIUC Theory CS Group†[1]

[1]Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA.

April 1, 2019

Abstract

This paper is an example of what happens when a bunch of theory students procrastinate.

1 Introduction

Since the introduction of the Turing machine by Alan Turing, there has been an uncountable amount of literature studying these machines. In addition, there are many variations of Turing machines and computational models, such as non-determinism, non-uniform computation and circuit complexity, and interactive proofs. In this paper, we introduce a new type of Turing machine—what we call a Time Traveling Turing machine, also known as a TM™. We note that the notion of time travel in computing has been studied before (of course in a much more formal and rigorous setting) [1].

In the following sections we: (i) formally define such a time traveling machine, (ii) define a new collection of complexity classes, whose languages are recognized by Time Traveling Turing machines, and (iii) relate these classes to classical complexity classes.

2 Motivation

On the eve of October 31st **2**018, Alan Turing, then 27 years of age, was sitting in a dimly lit room, scrawling many equations on various pieces of paper around his room. He sat back in his chair as he tried to slowly verify his calculations in excitement. He had just invented time travel. What he called a Time Traveling Turing machine. In his haste, he quickly built the machine the next day, November 1st. As he stepped into the machine, he began to feel the machine shake and rumble. In the last second before the time travel began, the machine code suffered from a segfault, and Turing traveling back in time to September 1st, 1939. Turing soon after failed to build a machine which would send him *back to the future*, and was stuck in this period of time forever. Of course, as we now know, Turing was soon recruited to Bletchly Park. The technology at the time was not powerful enough to build any sort of sophisticated Turing machine. His discovery of time travel was lost, and whenever he tried to share his discoveries after the war, people simply did not listen, dismissing his ideas as infeasible. Turing realized he would never return back to **2**01**8.**

*{rgandre2, mfjones2, plin15}@illinois.edu

†https://publish.illinois.edu/theory-cs/

A few months after this event, unbeknownst to all, some students from UIUC stumbled upon scraps of Turing's calculations on that spooky evening of 2018. In this paper we attempt to relay to you some of the ideas and proofs that Turing had.

3 Formal definition

A Time Traveling Turing machine, or to introduce as much ambiguity as possible, abbreviated as TM, is a regular deterministic Turing machine with the following additional feature: At any point during computation, the TM can enter a special time travel state $\circlearrowleft$. On entering this state, the machine is allowed to write $a(n)$ bits (where n is the length of the original input) onto a special *advice* tape. The machine then teleports back in time, restarting computation from scratch: it begins in the start state, and the original input of length n is on the work tape. The key difference is that through the magic of time travel, the TM now has access to an additional advice tape with $a(n)$ bits, which was sent into the past mysteriously from its future self.

Formally, a language L belongs to the class $\mathsf{TIMEYWIMEY}[a(n), t(n)]$ if there exists a Time Traveling Turing machine TM such that on input x:

1. After at most $t(n)$ steps from the start state, the machine either halts (and correctly decides if $x \in L$ or $x \notin L$) or must travel back in time by entering the time travel state $\circlearrowleft$.

2. Each time the TM time travels back, it can write at most $a(n)$ bits of information onto the advice tape for its past self.

Note that there is no restriction on the *number* of times a TM can travel back in time, only on *how much* information can be sent back to its past self.

4 Results

In this section we show how the class of languages recognized by Time Traveling Turing machines with $a(n)$ advice and $t(n)$ time relate to more traditional time and space-complexity classes. We also show that both adding randomness and non-determinism does not give TM accepting a language in $\mathsf{TIMEYWIMEY}[\mathrm{poly}(n), \mathrm{poly}(n)]$ more computational power.

Theorem 1. Let $a : \mathbb{N} \to \mathbb{N}$ and $t : \mathbb{N} \to \mathbb{N}$ be time-constructible functions. Then,

$$\mathsf{TIMEYWIMEY}[a(n), t(n)] \subseteq \mathsf{SPACE}[a(n) + t(n)] \subseteq \mathsf{TIMEYWIMEY}[a(n) + t(n), a(n) + t(n) + O(1)] .$$

Proof. We begin by proving the first containment. Let $L \in \mathsf{TIMEYWIMEY}[a(n), t(n)]$ be an arbitrary language and N be a TM deciding L. We design a regular Turing machine M deciding L using $a(n) + t(n)$ space. The machine M starts by copying the input onto a new work tape. Then M simulates N as normal. Any time N enters the state $\circlearrowleft$, M writes the advice tape onto its work tape according to N, M then clears the contents of its work tape (except for the advice string). Finally, M writes the contents of the original input back onto the work tape, along with the advice string, and resumes computation of N as if a time travel step had occurred. Clearly M only ever uses at most $a(n) + t(n)$ space, since the length of the advice tape is $a(n)$, and N only ever writes at most $t(n)$ bits onto its tape before entering $\circlearrowleft$.

For the second containment, let $L \in \mathsf{SPACE}[a(n) + t(n)]$ and M be a regular Turing machine deciding L in space $a(n)+t(n)$. Consider the following TM N. On input x, N simulates M for $a(n)+t(n)$ steps. It then writes the contents of its tape (which is of size $a(n)+t(n)$), current position of the head, and the state of M onto the advice tape (which requires an additional $O(1)$ space, where the hidden constant depends on M). Then N invokes the special state $\circlearrowleft$ to teleport back to its original state, now with access to its advice tape. If the advice tape is non-empty, N copies the advice tape onto the work tape, and restores its configuration. It then continues simulating M in this way until M halts and return an answer. Thus $L \in \mathsf{TIMEYWIMEY}[a(n) + t(n), a(n) + t(n) + O(1)]$. □

We can obtain the following corollaries by setting parameters in Theorem 1.

Corollary 2. $\mathsf{TIMEYWIMEY}[\text{poly}(n), \text{poly}(n)] := \bigcup_{c\in\mathbb{N}} \mathsf{TIMEYWIMEY}[n^c, n^c] = \mathsf{PSPACE}$.[1]

Corollary 3. $\mathsf{TIMEYWIMEY}[O(\log(n)), O(\log(n))] = \mathsf{L}$.

We now show the relation between $\mathsf{TIMEYWIMEY}[a(n), t(n)]$ and time-complexity classes.

Theorem 4. Let $a : \mathbb{N} \to \mathbb{N}$ and $t : \mathbb{N} \to \mathbb{N}$ be time-constructible functions. Then,

1. $\mathsf{TIME}[t(n)] \subseteq \mathsf{TIMEYWIMEY}[0, t(n)]$, and
2. $\mathsf{TIMEYWIMEY}[a(n), t(n)] \subseteq \mathsf{TIME}\left[2^{O(a(n))}t(n)\right]$.

Proof. The first statement follows easily from the definitions, since any Time Traveling Turing machine can choose to never time travel.

As for the second statement, let $L \in \mathsf{TIMEYWIMEY}[a(n), t(n)]$ and let N be a TM deciding L. We can simulate N by a standard Turing machine M that exactly mimics the behavior of N. The machine N runs for at most $t(n)$ steps before time traveling, so to show M runs in the desired time, all we need to do is show that N travels back in time at most $2^{O(a(n))}$ times.

Fix some string x as input to N. Consider the set of possible advice tape contents, which are (without loss of generality) strings in $\{0, 1, \Box\}^{a(n)}$ where $\Box$ is the blank symbol. We can build a directed graph G_x on this set by adding an edge from w to z if, starting on input x and advice contents w, the advice tape contains z when N next enters the $\circlearrowleft$ state. As N is deterministic, every vertex of G_x has outdegree at most 1. Moreover, since N must halt, the subgraph reachable from $\Box^{a(n)}$ (i.e., the initial advice tape contents) must be acyclic, as otherwise N would enter an infinite loop. This implies the subgraph reachable from $\Box^{a(n)}$ is a path of size at most $3^{a(n)} = 2^{O(a(n))}$. Thus N travels back in time at most $2^{O(a(n))}$ times, so M runs in $2^{O(a(n))}t(n)$ time total. The result follows. □

One immediate corollary of the above Theorem is the following characterization of the class P.

Corollary 5. $\mathsf{P} \subseteq \mathsf{TIMEYWIMEY}[0, \text{poly}(n)] \subseteq \mathsf{TIMEYWIMEY}[O(\log n), \text{poly}(n)] \subseteq \mathsf{P}$.

[1] Additionally, it is well known that $\mathsf{IP} = \mathsf{PSPACE}$. Therefore $\mathsf{TIMEYWIMEY}[\text{poly}(n), \text{poly}(n)] = \mathsf{IP}$. If the reader is still wondering why our model of computation should be taken somewhat-seriously, we remark that it is no less ridiculous than assuming access to an "all-powerful" prover, as in the IP model.

4.1 Does adding randomness give any more power?

Suppose a TM recognizing a language $L \in \mathsf{TIMEYWIMEY}[\text{poly}(n), \text{poly}(n)]$ now also had access to a random string of r with $|r| = \text{poly}(n)$ bits. Specifically, every time the machine travels back in time, it receives *the same* set of random bits r as it did in all previous time periods. Put differently, it does not get a new sequence of r random bits every time it travels back. We say that such a random TM N accepts a language L, if for all $x \in L$, N accepts x with probability at least $2/3$ over the random choice of r. If $x \notin L$, N accepts x with probability at most $1/3$.

Theorem 6. Suppose N is a randomized TM with access to a random string r of $\text{poly}(n)$ bits. Then every such TM can be simulated by a deterministic TM M.

Proof. The idea is that M will maintain two pieces of information on its advice tape at all times: (i) the current guess of the random string r, (ii) the number times the machine N has accepted a string x when simulated with the string r as its "random" input. Both of these strings are of $\text{poly}(n)$ size. After M has simulated N on all possible choices for r, it then outputs whether or not x was accepted by N by counting the number of times N accepted x with the string r fed to N by M. □

4.2 What about Non-determinism?

A language L is in $\mathsf{NTIMEYWIMEY}[a(n), t(n)]$ if there exists a non-deterministic TM M which always writes at most $a(n)$ bits on its advice tape, and between time periods, can only execute, non-deterministically, $t(n)$ of steps before time traveling back. Following Theorem 1, it is not hard to show that a similar Theorem holds if the Time Traveling Turing machine is allowed to act non-deterministically.

Lemma 7.

$$\mathsf{NTIMEYWIMEY}[a(n), t(n)] \subseteq \mathsf{NSPACE}[a(n) + t(n)] \subseteq \mathsf{NTIMEYWIMEY}[a(n) + t(n), a(n) + t(n) + O(1)] .$$

A consequence of Savitch's theorem is that $\mathsf{PSPACE} = \mathsf{NPSPACE}$. Hence via transitivity, we conclude that non-determinism does not buy you any extra power when you have access to time travel.

Corollary 8. $\mathsf{TIMEYWIMEY}[\text{poly}(n), \text{poly}(n)] = \mathsf{NTIMEYWIMEY}[\text{poly}(n), \text{poly}(n)]$.

4.3 Ok, what about if the TM *also* has access to some kind of oracle?

"Hmm...", the authors think slowly about this for a few seconds. After what seemed like an excruciating amount of silence, one of us blurts out thoughtlessly "Wait, what's that over there?! Look behind you!" As you turn around, we quickly hop into our DeLorean and zoom away.

5 Open problems

As pointed out, it is open whether or not having access to an oracle buys you any additional computational power. Additionally, does this newly introduced class $\mathsf{TIMEYWIMEY}[a(n), t(n)]$ provide other characterizations of well-known complexity classes? For example, is there a set of parameters for which this class is NL? NP? PH?

Acknowledgements. We'd like to thank the many PhD students within the CS Theory group at UIUC. There were many fruitful discussions about Time Traveling Turing machines, which were a great way to procrastinate on research, grading, preparing for labs, research, doing homework, research, and research.

References

[1] Scott Aaronson and John Watrous. "Closed Timelike Curves Make Quantum and Classical Computing Equivalent". In: *CoRR* abs/0808.2669 (2008). URL: http://arxiv.org/abs/0808.2669.

CONFIDENTIAL COMMITTEE MATERIALS

SIGBOVIK 2019 Paper Review

Paper 23: Need more RAM? Just invent time travel!

"Anonymous" Reviewers
Rating: Unjustifiably strong reject
Confidence: Expert (it's always the expert)

While this paper makes promising strides in the area of time-traveling complexity theory, we simply cannot excuse the fact that the authors don't include a citation to the work of Chapman, Katz and Muller [1] (no relation to the reviewers). While the present submission differs from the prior work by being in a different field of computer science, making better pop-culture references and arguably involving actual scientific merit, the idea of writing a SIGBOVIK paper about time traveling machines is clearly not novel and therefore it is our entirely objective opinion that the submission should be rejected, or at least published alongside this review so everyone can be reminded of how insightful ~~we~~ the prior authors are.

[1] Peter Chapman, Deby Katz and Stefan Muller. A Proposal for Overhead-Free Dependency Management with Temporally Distributed Virtualization. SIGBOVIK 2013.

WICCAN: (deep) learning directly from the future

Amanda Coston, Alan Mishler

Abstract

Deep learning methods are extremely popular but suffer from a number of limitations, including computational and conceptual complexity, fragility to input variation, and poor generalizability. Perhaps most worryingly, they rely on data from the past to train models and generate predictions about the future, raising questions about the validity of their output.

The dark arts offer potential solutions to a number of these issues. Deep learning currently relies on a handful of alchemical techniques, but it has yet to take advantage of the full array of available magical methods. Here, we propose a novel type of deep learning: Weakly Independent Concurrent Convolutional Adversarial Networks (WICCAN). WICCAN eschews the reliance on the past that characterizes other techniques. It is model-free, data-free, and space-time-free, and can predict unseen or unrealized labels, outcomes, and events. WICCAN performs comparably to the state of the dark art, with the advantage that it runs in O($\emptyset$) time and does not require access to deceased prophets at runtime.

1 Introduction

1.1 Background

Deep learning methods are extremely popular and perform at state-of-the-art in a wide range of prediction and labeling tasks. However, they suffer from a number of well known limitations. First, they typically require very large amounts of training data, which may be expensive or impractical to acquire and annotate. Second, they can be fragile; for example, a change in a few pixels can cause a convolutional neural net to fail to correctly label an image. Finally, they can be extremely computationally expensive to train and store.

Deep learning suffers from two additional limitations that have been underappreciated in the literature. First, like other machine learning methods, deep learning overwhelmingly relies on data from the past to train models and generate predictions about the future. This approach requires that the data-generating mechanism remain stable over time, an unrealistically strong assumption since a lot of things happened in the past which might not happen again. Furthermore, something new might happen.

Second and relatedly, deep learning models only generalize to the type of data that they learned from. For example, a convolutional neural net trained on images of animals is unlikely to perform well when asked to predict the location and timing of future earthquakes.

Human prophecy has a nearly complementary set of strengths and limitations. Like deep learning, prophecy also aims to uncover the unseen and predict the future. Unlike deep learning, however, prophecy is not fragile and does not require training data; in fact, it has proven remarkably robust to the presence of data. This means that training time is not constrained by the size of the data and that it generalizes equally way to nearly anything. Finally, prophecy takes the sensible approach of peering directly into the future, rather than the somewhat backward approach of looking into the past in order to make guesses about the future.

Although deep learning and the dark arts are often invoked together, there has been surprisingly little discourse between researchers in the two fields. Deep learning has traditionally utilized only a few basic alchemical techniques, leaving untouched a wide array of other methods in the dark arts.

1.2 WICCAN: A new type of neural net

Here, we propose a new method that builds on existing areas of overlap while drawing on additional strengths of the two approaches. The method is called Weakly Independent Concurrent Convolutional Adversarial Networks (WICCAN). WICCAN predicts unseen and/or as-yet-unrealized labels, events, and facts, while eschewing the reliance on data that characterizes other methods. WICCAN is therefore immune to overfitting and generalizes perfectly to any type of task, even when the future does not resemble the past. Notably, WICCAN does not require any pretraining or parameter tuning; the choice of a familiar is all that is required at both train and test time.

We empirically evaluate our method on standard benchmarks as well as new tasks that we believe to be more representative of real-world problems. Our method performs comparably to the state of the dark art, with the advantage that it runs in O($\emptyset$) time and does not require access to deceased prophets at runtime.

The remainder of this paper is organized as follows: In section 2, we summarize relevant work in deep learning and the dark arts. Section 3 details the procedure, to the extent allowable by the Ardanes. Section 4 situates this work within the recently explosive literature on fairness. Section 5 presents our empirical results, which show that WICCAN outperforms existing methods on both benchmarks and unseen real world prediction tasks. Finally, section 6 uses our method to propose future work. Sections 3 and 6 are written in the grimoire tradition and may be incomprehensible to mortals.[1]

2 Related Work

Both machine learning and the dark arts have witnessed significant progress in the past centuries, and this paper builds on the formidable literature of both disciplines. Machines are now outperforming humans on vision tasks and games like Jeopardy and Go [1] [2]. Horcruxes are now able to achieve immortality, and wielders of white magic have demonstrated that a patronus can defeat Dementors [3] [4]. The literature in both fields is too vast to properly review; in this section we discuss a select few relevant works.

In a seminal vision paper, researchers from the University of Toronto showed that deep convolutional neural networks (DCNN's) vastly outperformed other methods on ImageNet classification [5]. The architecture consists of five convolutional layers as well as several fully connected and max pooling layers. The authors proposed methods to reduce the risk of overfitting and to speed up training. While similar in spirit to DCNN's, our method requires only an arbitrary number of layers, trains in sub-constant time, and has no risk of overfitting. ImageNet led to an explosion of followup work, with deep learning researchers investing considerable effort on designing architectures and appropriate activation functions and on developing methods to reduce training time. We believe our method will considerably advance the field since it requires no specification of an architecture and runs in near-trivial time, which we denote as "time-free."

Our method also builds upon the recent work of Nostradamus, whose method of foresight outperformed nearly all political pundits in predicting the 2016 election of Donald Trump [6]. Notably, Donald Trump had never before been elected, so many existing methods were unable to predict this "black swan" event. However, our method significantly improves upon Nostradmus's prophecies, which are characteristically vague albeit never incorrect. Our method inherits his perfect accuracy while specifying the details of the event in question, which aids in the interpretability of the model.

Previous work in the prestigious SIGBOVIK conference considered whether parapsychology can be used to influence people's minds, ultimately and unfortunately finding that the author was "supernaturally unpersuasive" [7]. Our method also concerns the inner workings of mortal minds, with an emphasis on reading versus influencing them, which we find to be a more tractable problem.

[1] If you have the misfortune of being mortal, we recommend reading these sections upside down, and we offer our condolences in the likely event that you are turned into a bat.

(a) My friend's dog

(b) My dog

Figure 1: The optimal familiars for EEVEE

Our method is also relevant to future work in machine learning, which will use more layers and more activation functions to accomplish more things. Perhaps the most relevant of these future methods is EEVEE, Empirical Evidence Variational Expert Encodings [8]. EEVEE uses a familiar-stacking architecture and finds that Tiggy and Eevee, who are displayed in Figure 1 are optimal familiars and are incidentally optimally cute.

As far as the authors are aware, we are the first to propose a paradigm that leverages the benefits of both machine learning and sorcery.

3 Methods

WICCAN consists of an input layer, an arbitrary number of medial layers, and an output layer. The network is trained using SGD, as described in Algorithm 1. Unlike previous methods, training does not require data, which must be necessarily collected in the past relative to training time and which therefore bears only a tenuous relationship to the post-training future.

WICCAN is also distinct from other neural nets in several other respects:

- The input layer can accommodate any size or type of input, as long as it can be framed as a statistical estimation task or as a space-time-based query, such as "will this paper be published in a prestigious proceedings?" or "will a locust swarm devastate the coastal croplands?"
- Unlike other neural nets, WICCAN does not distinguish between hidden and non-hidden layers; all layers in WICCAN, including the output layer, are hidden from mortal sight.
- The only hyperparameter which needs to be selected is the choice of familiar, e.g. cat, owl, snake, or undergraduate research assistant. Other hyperparameters such as the learning rate, learning rate schedule, mini-batch size, paranormality, momentum, convolutional stride, regularization coefficients, broom length, and the number and structure of the medial layers are not explicitly specified and do not need to be separately learned. All parameters, hyperparameters, pseudoparameters, supernatural parameters, and non-parameters are learned simultaneously in a single burst of power.
- Early stopping: It is extremely dangerous to stop the training spell early.

WICCAN is trained until training is complete, at which point it can be used to predict any unseen or as-yet-unrealized labels, events, or facts.

3.0.1 Complexity

WICCAN retains the full complexity and mystery of the natural world in which it operates. Since no data is used in training, the training time is O(1), although the associated constant is unknown and unknowable. Runtime is O($\emptyset$): answers to queries are instantly available.

```
Algorithm 1: SGD: Spell-based General Divination
for number of incantations $k$ do
    update the network by intoning the following:
        $\delta_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right] \quad (1)$
end
Note that $k$ is not set prior to training but is revealed over the course of training.
```

Table 1: MNIST empirical evaluation

Method	Test accuracy
WICCAN	Slightly above perfect
Support vector machines	0.994
DCNN	0.997
EEVEE	1.00
k-Nearest Neighbors	0.994
Asking my cat	0.996

4 Fairness

In most machine learning contexts, the test set is used as a proxy for the future, yet machine learning researchers regard it as “unfair” to use data from the test set to train the network. We find this stance strange, since the future is ultimately the object of interest. In fact, we regard it as substantially more unfair to use data from the past to make claims about a possibly entirely different future.

We also think pejorative references to deep learning as “alchemy” are unfair. Both alchemy and deep learning have seen widespread successful application [9, 10], despite having their share of misses [11, 12].

5 Empirical Results

We demonstrate the performance of our methods empirically, on a standard machine learning benchmark and on two new prediction tasks which we believe are more representative of real-world tasks. We compare these to current and future state of the art methods.

MNIST is a dataset of handwritten digits where the prediction task is to classify each digit as one of 0-9 [13]. MNIST is used as a standard benchmark for evaluating machine learning algorithms. Table 1 shows that WICCAN outperforms all other methods on MNIST, achieving over 100% accuracy. The authors note that not only was our method able to classify the number correctly but it was also able to reconstruct the users’ thoughts as they wrote the digit, a significant improvement upon existing methods. For exposition we include a few excerpts in Table 2.

Motivated by real world challenges, we introduce three new prediction tasks to the literature: predicting 1) Bigfoot sightings, 2) who will dance together at the Yule Ball, and 3) next year’s holidays. Understanding the location and movement of Bigfoot is important for maintaining the ecological stability and balance of the forests of the Pacific Northwest. Ecologists, who have asked for years for better prediction of Bigfoot, have extolled our method as telling them *"exactly what they didn’t need to know"* and providing *"unvaluable information"* about Bigfoot’s whereabouts. Our method was able to predict 14 Bigfoot sightings in the next 1000 years, which compares to 0 predicted by current methods like logistic regression, reputable newspaper articles, and science.

The Yule Ball is a major social event at Hogwarts that can determine the course of one’s romantic and social trajectory for the immediate future (on the order of hours). Wizard behavior is notoriously difficult to characterize or predict, but yet our method is able to not only predict the couples who will dance together but also is able to itself dance an enchanting foxtrot. While DCNN’s are able to achieve similar results, our method’s runtime vastly outperforms DCNN’s, which takes 10 lifetimes to run.

Table 2: MNIST mind exposition

Thought	Digit Written
I used to be 8 once	8
6 doesn't really deserve to be a number...kinda creepy	6
Did I meet that guy at the dentist last year?	0
My handwriting is so pretty	4
Why did they tell me to write 5?	6

Table 3: WICCAN prediction for holiday dates in 2020

Holiday	Prediction
Valentine's Day	February 14
April 1st	April 2nd
Halloween	Every day
Someone's Birthday	October 6
Fourth of July	July 4th

Knowledge of the dates of future holidays like Thanksgiving and Valentine's Day is important to most people who want to celebrate with their friends and families. A significant challenge to holiday planning is the uncertainty about when holidays will be. Our method is able to predict some future holidays for 2020, which are demonstrated in Table 3.

6 Conclusion and Future Work

WICCAN will be used in the future to predict future work.

Acknowledgments

We would like to acknowledge the mere mortals who will give us invaluable advice in the distant future: Duncan McElfresh, Anjalie Fields, Kevin Lin, Lily Potter, Channing Huang, Riccardo Fogliato, Benjamin LeRoy, Nic Dalmasso, Kwangho Kim, and Siddharth Ancha. We are also grateful to the immortals who will read this paper and think about it for a little bit, which include Lisa Lee and God. This research is sponsored by the Graduate Research Fellowship from the Ministry of Magic.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[2] D. A. Ferrucci, "Introduction to "this is watson"," *IBM Journal of Research and Development*, vol. 56, no. 3.4, pp. 1–1, 2012.

[3] J. K. Rowling, *Harry Potter and the deathly hallows*, vol. 7. Bloomsbury Publishing, 2013.

[4] J. K. Rowling, *Harry Potter and the prisoner of Azkaban*. Bloomsbury publishing, 2015.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[6] M. Nostradamus, *Les propheties*. Fayard/Mille et une nuits, 1998.

[7] D. Edelstein, "This grad student studied parapsychology — and you won't believe what he found!," in *SIGBOVIK*, 2018.

[8] L. Lee, "Empirical evidence variational expert encodings: A better method," in *International Conference on Machine Learning*, pp. 100–1105, 2021.

[9] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," pp. 1–9, 2014.

[10] J. K. Rowling, *Harry Potter and the sorcerer's stone*. Bloomsbury publishing, 2013.

[11] A. Ghorbani, A. Abid, and J. Zou, “Interpretation of Neural Networks is Fragile,” no. Lipton 2016, 2017.

[12] “Chinese alchemical elixir poisoning,” *Wikipedia*, accessed 03/01/2019.

[13] Y. LeCun, C. Cortes, and C. Burges, “Mnist handwritten digit database,” *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, p. 18, 2010.

On CLI-based Renderers:
In which we investigate the utility of rendering teapots in a command line

Michael Sandler

University of Illinois at Urbana-Champaign

March 16, 2019

Abstract

We investigate the feasibility of producing a renderer within a CLI environment, and the usability of thereof. We also examine the effects that a blissful disregard for good coding practice and performance can have on such a project. A proof of concept is completed in the form of a cube, and extended in order to produce a teapot.

1 Introduction and Motivation

Rendering can often be highly technical and sophisticated, and cutting-edge technologies are often created to speed it up. We propose an alternative, in which we use rendering to actually *set technology back* a few years. In pursuit of this, we wrote a renderer that produces nothing except pictures of teapots. While there are technically controls, the refresh rate is so slow that the rendering could hardly be called realtime, even if we used more sophisticated techniques. For this reason, we also do not pursue fast rendering speeds.

The reason for teapots is simple: It is common knowledge that the presence of teapots has a positive impact on both the mental and academic well-being of students. It is less known, and a worthy research topic, as to the effectiveness of *virtual* teapots. However, as virtual teapots produced in industry-standard renderers are utterly indistinguishable from real teapots, this experiment would not reveal anything interesting. So, in order to investigate this, we wrote a 3d renderer in python that renders a teapot.

2 Implementation

Although the base code was a barebones implementation of a raycaster, we nevertheless encountered a few pitfalls in in implementaton.

1. This project was started less than twelve hours before my analysis final for which I had not studied. This was not a good idea, and I do not recommend that people follow my example.

2. It was shockingly hard to convince people that 2 was a photo of a pixellated cube, and not an amorphous blob.

3. The code was written in python, a language known for its high speed and efficiency.

4. Refusing to use external libraries (potentially with C bindings) may also have contributed to the utter lack of rendering speed.

That said, the technical details of the implementation were as follows:

- Renders were produced under diffuse lighting from a directional source.

- The rendering algorithm was a basic raycasting algorithm, casting rays per-pixel and checking for triangle collision.

- No spatial splitting algorithm was used, as the authors felt incompetent and lazy.

3 Results

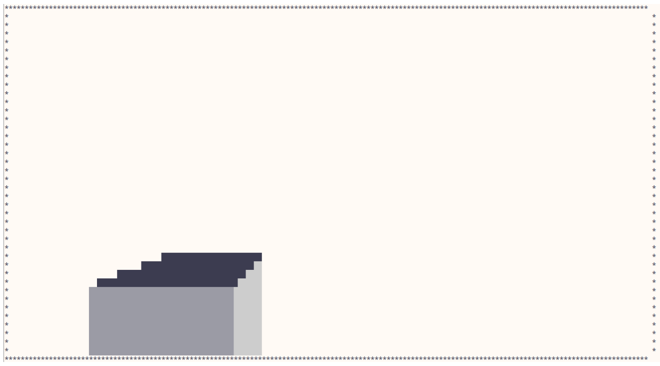


Figure 1: The initial render of a cube.

Surprisingly, figure 2 only took a few milliseconds to render. However, the ncurses overhead alone meant (surprisingly enough) that the cube still had non-negligible refresh rates. Although there were realtime capabilities for camera movement, our guinea pigs remained unconvinced as to the fact that a cube was indeed being rendered, and not some bizarre-looking square contraption.

Frustrated, we plugged in an STL model of the teapot. Due to the issues detailed earlier, this render took half an hour; however, it was far more successful in convincing cynical spectators that our renderer was indeed effective and creating pictures of unphysically lit triangles.

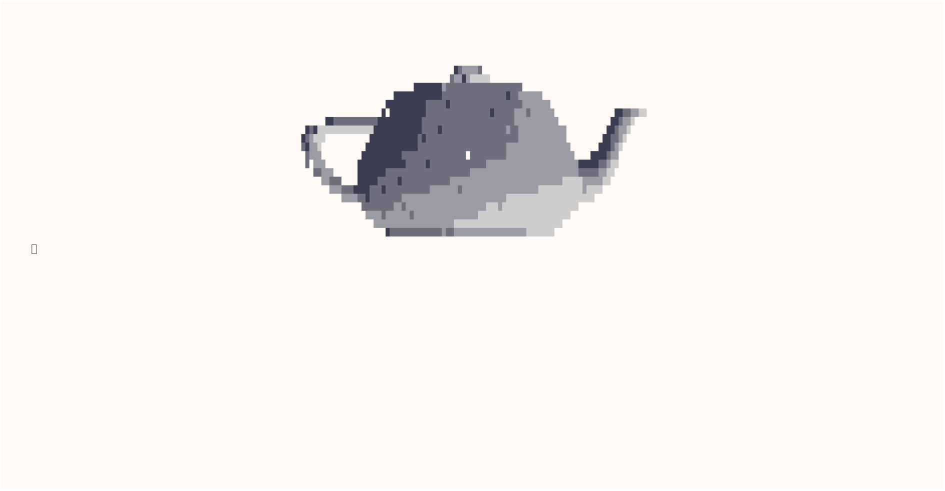


Figure 2: The best-looking teapot render.

A cursor is visible due to a slow and ineffective rendering algorithm.

4 Conclusion

This project led the authors to a single conclusion: Renderers, CLI or not, should not be written in python. A preliminary port to C++ yielded speed improvements of three thousand percent. However, the idea of a working CLI renderer is not wholly ridiculous, and can lead to effective (and amusing) results.

As a direction of further investigation, we intend to bring near-realtime rendering to the command line, and allow command line, modal (in the sense of Vi) editing of models. We hope that a keyboard driven editor with no mouse support will convince others as to the complete uselesness of our work.

AVOIDANCE OF EXTRATERRESTRIAL
INTELLIGENCE INSTITUTE **Oscar Hernandez**

Oscar Hernandez
AETI Institute
189 Bernardo Ave, Suite 200
Mountain View, CA 94043, USA
Phone: +1 (646) 961-3715
E-mail: ohernandez13@simons-rock.edu
URL: `https://mathemonads.github.io`

π, 2019

President John F. Kennedy
The White House
1600 Pennsylvania Ave NW
Washington, DC 20500
United States of America

To Mr. President or whomever it may concern,

We saw your speech on C-SPAN, where you declared interest in exploring space. But I'm warning you – if you do that, people will be angry, and want to shoot you. That's not a threat, I actually love and respect you very much, sir.

All joking aside, we are practical learned men. Instead of spending money on an extravagant wild goose chase, you should invest in local communities: fix the welfare state, end the war on drugs, and ensure that people of color have those same rights as everyone else.

You worship British explorer George Mallary for climbing Mt. Everest "because it was there". By that logic, we should eat cake "because it's there": an understandable desire that causes heart failure in excess. Not to mention the fact that he DIED climbing it.

We've seen what unwarranted exploration leads to: the wild wreckage that Christopher Columbus brought to the Americas and the massive deforestation in the Amazon, to name a few.

So long as things "are there", we will continue to explore, harvest and destroy until things "are no more". *Veni vidi vici. Quia oblitus sum dicere integros communitates viam?*

I urge you Mr. President to stop this egotistical quest of stepping on what you have not stepped on. There is beauty and magic in the unknown that dies when it is discovered. Once the magic dies, it becomes a resource ready to be consumed. And I could do without the moon cheese Mr. President; cow cheese oughta' do it alright alright alright.

Really, we should just have the common decency to leave the aliens to their own planet, as they have paid us that respect for millions of years. #yeswecandotherightthingandnotgotospace

Sincerely,

Oscar Hernandez
American Citizen, Responsible
Voter, Need I go on?

Pop culture

Their full results are posted at: `https://tinyurl.com/bracketiest`

Which ITG Stepcharts are Bracket-Jumpiest?: In Which They Milk the 「A Boring Follow-Up Paper to “Which ITG Stepcharts are Turniest?” Titled, “Which ITG Stepcharts are Crossoveriest and/or Footswitchiest?”」 Series for All Its Worth in Publication Count After All, or: Hit Me With An Encore

Ben Blum

bblum@alumni.cmu.edu

Abstract

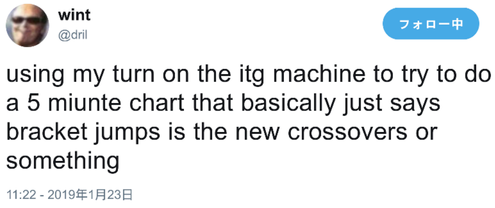
In which I break last last year's promise of no future work.

Categories and Subject Descriptors D.D.R. [*Exercise and Fitness*]: Arcade Dance Games

Keywords bracket, groove, in, jumps, the

1. Introduction

Recent work by (dril 2019) proposed the hypothesis that recent stepchart authors have grown bored with the array of technical ITG step patterns documented to date (Blum 2016, 2017), and have moved on to break the old model's one-foot-per-arrow assumption to allow for even more technical patterns yet. I paraphrase their main conclusion as follows:


wint
@dril
フォロー中

using my turn on the itg machine to try to do a 5 miunte chart that basically just says bracket jumps is the new crossovers or something

11:22 - 2019年1月23日

All good researchers know that when rise the standards for software or hardware performance (or stepchart trickiness, as the case may be), they must revisit their own work to prove its ongoing relevance to the research (dance) community at large. Thus I must regrettably break the promise ~~I~~ the authors set forth in (Blum 2017) (see Figure 1) and revisit ~~my~~ their old future work section (XXX: they said to change any first-person “our prior work” stuff like this for double-blind review but like cmon this makes no sense? theyll totally see through this (TODO: maybe email the PC chair for advice? (FIXME: make sure to remove these comments before the camera-ready deadline!!))), extending it to handle these new so-called “bracket” jumps.

A Boring Follow-Up Paper to “Which ITG Stepcharts are Turniest?” Titled, “Which ITG Stepcharts are Crossoveriest and/or Footswitchiest?”

Ben Blum

bblum@cs.cmu.edu

Abstract

In which I deliver on last year's promise of future work.

Categories and Subject Descriptors D.D.R. [*Exercise and Fitness*]: Arcade Dance Games

Keywords crossovers, footswitches, jacks, sidefoots

1. Introduction

Let's resume right where I left off in my last paper (Blum 2016), shown in Figure 1. Unlike mainstream conferences, SIGBOVIK doesn't make me waste space repeating all the background material, and I can just say go read that paper first and get back to me. It's probably a lot funnier than this one anyway, which is gonna be sort of dry, and really of interest only to other ITG players who already know what's going on.

The TL;DR is that I made a program which figures out how to foot stepcharts in the least crossovery possible way (short of double-stepping everything), then found which charts ultimately had the most. The algorithm also naturally identifies footswitches and jacks, and sometimes it's smarter than me in amusing ways. I put all the goodies in a giant spreadsheet at `http://tinyurl.com/crossoveriest`, and the program itself is of course freely available at `https://github.com/bblum/sigbovik/blob/master/itg/code/ITG.hs`.

2. Revisiting Turniness (Flashback Scene)

Recall Table 1 from the last paper, in which I left undefined the facings for LL, DD, UU, and RR, the four

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee, provided... honestly, provided nothing. The ACH is already flattered enough that you're even reading this notice. Copyrights for components of this work owned by others than ACH must be laughed at, then ignored. Abstracting with credit is permitted, but abstracting with cash is preferred. And please tip us, for the love of Turing.

SIGBOVIK '17 Pittsburgh, PA, USA
Copyright © 2017 held by owner/author(s). Publication rights licensed to ACH.
ACH ...$15.00

Figure 1. (okay twist your head to read this)

		Right foot			
		←	↓	↑	→
Left foot	←	**?**	UR	UL	U
	↓	DL	**?**	L	UL
	↑	DR	R	**?**	UR
	→	D	DR	DL	**?**

Table 1. Facing directions.

footswitches. I show a typical DD/UU footswitch pattern in Figure 2(a), and typical LL/RR switches (henceforth “crossover footswitches”) in Figure 2(b). To step these patterns, the player still alternates feet as usual, but must lift one foot off the repeated arrow before stepping it with her other foot. Chart authors will often, but not always, include a “mine cue” (shown in the figure) to hint that the second foot should switch onto the same arrow.

Figure 1. (yeah i reused this joke from last time ok deal)

SIGBOVIK '19 Pittsburgh, PA, USA

ACH ...$13.37

Their full results are posted at: https://tinyurl.com/bracketiest

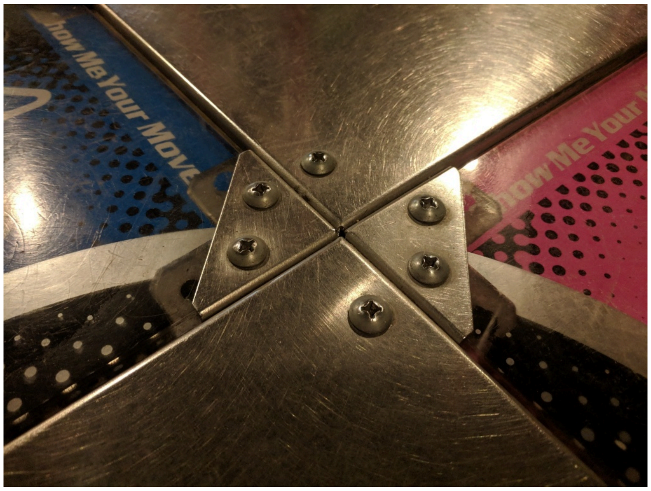


Figure 2. Detail of metal corner brackets, this paper's namesake. Photo credit JIM.

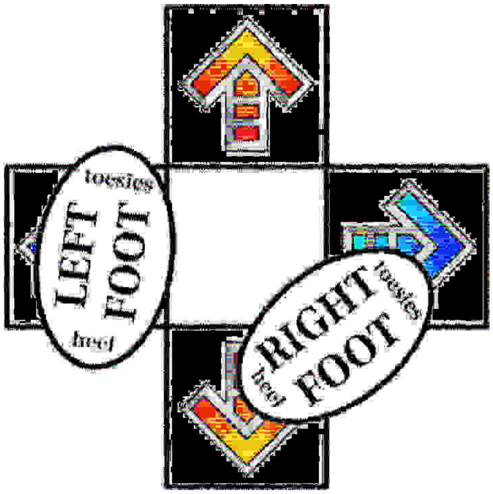


Figure 3. Down+right bracket-jump real-world example.

2. Overview

What, then, is a bracket-jump (which I shall *not*, henceforth, abbreviate for brevity)? Put simply, whenever two arrow-shaped obstacles proceed simultaneously towards the protagonist directional indicator targets (Blum 2016), while a novice player might think they must step with both feet at once, one for each arrow, experts often find it more convenient (i.e., less overall foot motion) to use whichever single foot is closer at the time to hit both arrows by triggering one arrow's sensor with the heel and the other with the little bitty toesies. Pads are typically constructed with a small triangular metal bracket at the corner of each arrow panel, as shown in Figure 2, which the bridge of the foot must cross to achieve this, hence the name "bracket jump". In case my prose explanation is not up to snuff, I also show in Figure 3 a high-quality graphics render of a player's typical foot positions during a down+right bracket-jump (henceforth "DR", et cetera).

The reader, or stepper, may notice the extreme angle of footing depicted in the latter figure, which is necessary to reliably trigger both pad sensors. Accordingly, this maneuver is comfortable (and hence preferable to a normal jump) only if the player is already facing in roughly the same direction (Blum 2016). Note also that the two "candle jumps", LR and DU, are not possible to bracket, unless your foot size is (physically) beyond the scope of this work. The next section will attempt to codify (ahem) when preceding patterns encourage the player to bracket rather than jump, and thereby identify how "bracket jumpy" each chart is.

3. Algorithm Design

I extended the crossoveriness et cetera algorithm from (Blum 2017) to reason about jumps, which previously it treated all identically, ignoring the arrows involved and allowing the player to reset her footing as desired. Now, it considers LD, LU, DR, and UR jumps as potentially bracketable, allowing the player to continue a stream of alternating feet uninterrupted therethrough.

Confession. When I was first brainstorming this project, I had some grand visions of unifying the turniness algorithm (Blum 2016)—which accounts for U/D steps to figure out how far the player must turn each step, but has no idea which way she is actually facing at any given time—with the crossoveriness one (cited just above; cmon how much do you want me to repeat these same two citations, gimme a break)—which totally ignores U/D steps and just figures out which arrows the left and right feet must each step—in some theoretically beautiful way to produce the unquestionably perfect footing sequence for each jump. However, as I was considering how to incorporate features from last time's algorithm, potentially allowing crossover brackets (Section 6.2) and footswitch brackets (Section 6.3), I realized that ultimately there would be no restrictions on what jumps were bracketable; the algorithm would oops simply twist and turn as much as necessary to bracket everything, and this paper would become just "Which ITG Stepcharts Have the Most Jumps?", and like who wants to read that.

One may think to simply try either bracketing each jump or not and seeing what combination gives the minimum turniness result, but since whether to bracket each jump or not affects subsequent steps' footing, each choice cannot in general be solved independently, and I hope you can see where this is going. Now, the last time I tried to solve an exponentially-sized problem, it took me 7 years (Blum 2018) (and I still ended up with a pile of heuristics after all anyway), so considering I started this project a week before the deadline, I opted instead to just code up a bunch of ad-hoc rules to handle all the different bracket jump patterns that occurred to me to write test cases for.

As before, the code is available at https://github.com/bblum/sigbovik/blob/master/itg/code/ITG.hs. To give a sense of how much its elegance has been despoiled since the last version: 62 lines of Haskell (which computed crossovers, footswitches, jacks, doublesteps, *and* crossover footswitches) has grown to 156 lines (just to handle this one. new. feature), and the once-simple datatype definition of

```
data Step = L | D | U | R | Jump
```

has become the unwieldy:

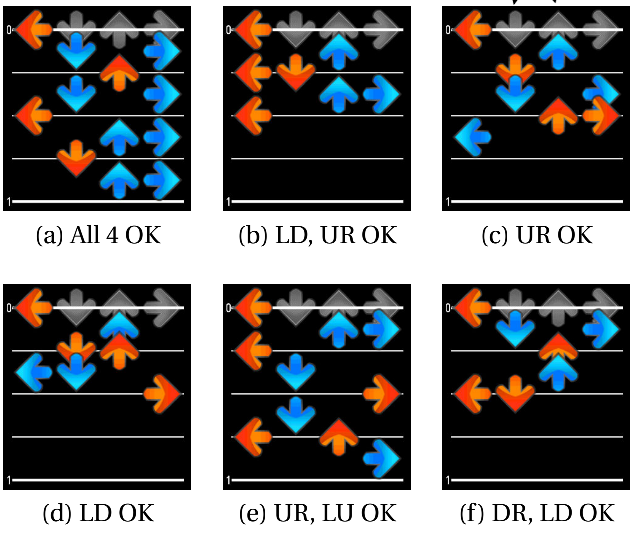
```
data Arrow = L | D | U | R
data Jump = LD | LU | DR | UR | LR | DU | Other
data Step = A Arrow | J Jump
data Foot = LeftFoot | RightFoot
```

The general gist is that in addition to tracking which foot stepped the last arrow and optimizing for alternating feet, we also track which arrow(s) each foot was last on, and whenever we encounter a LD/LU/DR/UR jump, bracket it if both:

1. The last foot is opposite the foot required for the jump (e.g. L for LD), and
2. The last foot's last note(s) does not intersect the arrows involved in the jump (i.e., R not on D preceding LD).

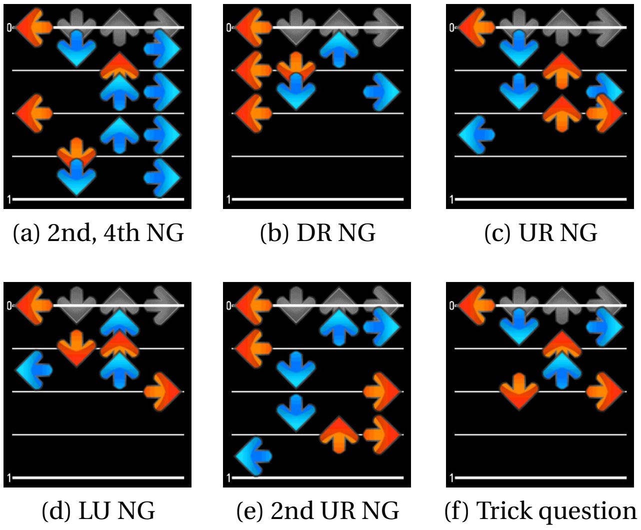
In lieu of a detailed algorithm listing, I'll simply show some of those most notable test cases in pictures and explain in prose how a player would typically step them, which was pretty much my implementation strategy to begin with.

Figure 4 shows a bunch of patterns with a bracketable jump, and each subfigure in Figure 5 shows a corresponding unbracketable case. Whenever two jumps are pictured in the same unit test, we are concerned with whether the latter one is bracketable. I'll now explain how the algorithm handles each case.


(a) All 4 OK (b) LD, UR OK (c) UR OK

(d) LD OK (e) UR, LU OK (f) DR, LD OK

Figure 4. Examples of bracketable jumps.


(a) 2nd, 4th NG (b) DR NG (c) UR NG

(d) LU NG (e) 2nd UR NG (f) Trick question

Figure 5. Examples of unbracketable jumps, each corresponding to the similar pattern in Figure 4.

a. In the OK case, the right foot can bracket every jump (condition 1), as the left foot is always on a different arrow than the jump (condition 2). In the NG case, condition 2 is violated for the second and fourth jumps.

b. Version of (a) testing that preceding bracket-jumps also contribute to condition 2. The NG case would be a footswitch bracket requiring rather uncomfortable footing angles.

c. In OK, DR cannot be bracketed, but clears the prior footings so UR can. In NG, LR can be bracketed, meaning UR cannot while still alternating feet (condition 1).

d. DU jumps can never be bracketed, but can be stepped facing either direction, and this must respect prior footings. In both cases DU is stepped with right foot U, which allows bracketing LD but not LU thereafter.

e. In OK, the crossover does not interfere with either bracket and the player can alternate feet throughout. In NG, either doublestepping or jumping normally is required (or a future-work crossover bracket; see Section 6.2).

f. In OK, the player easily footswitches on U. In the corresponding case, the player can actually just jack U instead, stepping it twice with her left foot before bracketing DR again with her right.

These examples are available as test-bracket-*.sm unit tests in the code repository. Hopefully armed with this understanding of the bracket jumping rules, let's now move on to real-world stepcharts.

4. Evaluation

The experimental corpus has grown even more since last time, now comprising 17340 stepcharts from 182 packs. Of note, all three Technical Showcase packs (collaborative packs where the community at large was encouraged to submit their freshest beats and judged on complexity) were released in the last two years, which have proved to be important sources of diverse bracket-jump patterns for this research. I would also have liked to re-rank the crossoveriest and footswitchiest charts from last time to include these submissions, but I frankly don't have time. The results spreadsheet of bracket jump counts (as well as more crossovers, footswitches, et cetera for charts newer than the last paper) is available for your browsing pleasure at https://tinyurl.com/bracketiest.

I pose two evaluation questions, to be answered in the following subsections respectively.

1. Which songs, packs, and/or step-cartographers are bracket-jumpiest?
2. Is bracket jumps the new crossovers or something?

4.1 Bracket-Jumpiness

I measured the overal bracketiness of each chart in two ways, first, by comparing the number of bracket jumps against the total number of steps, or the *bracket-jump step percentage* (BJS%); second, by comparing against the total number of jumps only, or the *bracket jump jump percentage* (BJJ%). The BJS% indicates a chart's overall density of steps requiring the player to step across the brackets (which depends on how jumpy the chart is to begin with), whereas the BJJ% measures perhaps the author's intentionality in patterning their jumps either as brackets, or as random where just some of them happen to be bracketable.

Table 1 shows the leaderboard for the former metric, and Table 2 the latter.[1] Note that the aforementioned recent Technical Showcase pack series put up three bracketiest charts in the BJJ% category; meanwhile, the UPS packs, known for their gimmicks and general lack of respect for player comfort, secure several top spots on the BJS% board. For comparison's sake, the overall BJJ% of the entire corpus is 19.8% (93k/470k), meaning that roughly 1 in 5 randomly-patterned jumps are by chance bracketable. The overall BJS% (a thoroughly less meaningful statistic) is 0.7%.

Ft.	Name	Pack	#BJ	BJS%
10	SOBA	Squeaky Beds &c.	366	83%
11	Get Off of My Way	UPS 3	115	28%
12	Hardware Store	Keyboard Coll. III	207	23%
13	Firestorm	BemaniBeats 3	153	31%
14	Bounce	UPS 2	126	21%
15	Ikaros Dynamite!!!!	UPS 2	238	34%
16	Mermaid Island	Tachyon Alpha	346	49%
17	Toccata & Fugue	CuoReNeRo M.P.	255	24%

Table 1. Charts of each difficulty with the highest bracket-jump density among all steps (BJS%).

Ft.	Name	Pack	#BJ	BJJ%
9	Drifting Away	UPS 4	26	88%
10	SOBA	Squeaky Beds &c.	366	99%
11	Save Miracles	ECFA 2019	42	95%
12	Electrical Paradise	Chic. Timing Auth.	35	100%
12	Encore	Tech. Showcase	29	100%
12	Nemeton	Subluminal	12	100%
13	Decadent Dandy	Tech. Showcase 3	12	100%
14	Nageki no Ki	Valex's M.4-A.A. 8	116	94%
15	Beach Party	Tech. Showcase 2	162	80%
16	Mermaid Island	Tachyon Alpha	346	79%
17	Zombie Sunset	Jummy Jawns 2	325	88%
69	koopa bling	UPS 4	39	98%

Table 2. Charts of each difficulty with the highest percentage of their total jumps bracketable (BJJ%). All participants of the 3-way tie for 12-footers are listed.

Next up is bracket jumpiness by pack and by author. Tables 3 and 4 show the top 10 packs in both bracketiness metrics, and Tables 5 and 6 the top 10 authors (filtered by having written at least 10 charts). For the packs, I also list the release year of each; note how despite the relatively even spread of years in BJS%, 2018 dominates the BJJ% leaderboard, suggesting that older charts' bracket-jumps arose by chance simply from having a lot of jumps to begin with. I'll come back to this point in Section 4.2. In honorable mentionth place is chart author Halogen–, whose 11 charts contain 154 jumps, *none* of which are bracketable. What a purist!

4.2 Historical Trends

I sought to prove (dril 2019)'s claim that bracket-jumps' popularity has skyrocketed in recent years of stepchart-making, by finding the progression of bracketiness across past years of stepchart packs. However, actually assigning a firm release date to every pack on my hard drive turned out to be a feat of internet archaeology unto itself, even involving archive.org for one step. I'll spare you the details, but you can peruse them in the second spreadsheet of the dataset linked at the start of this section. Ultimately, all packs older than 2013 had to be grouped together, as no

[1] When measuring BJJ% I excluded charts with fewer than 10 jumps in total, which put up a handful of false positives in the 100% ranks.

Their full results are posted at: `https://tinyurl.com/bracketiest`

Pack name	Year	#charts	BJS%
Keyboard Collaboration III	≤2012	17	12.4%
Squeaky Beds and Leaky Faucets	2018	125	4.9%
Keyboard Collaboration I	≤2012	12	3.6%
CuoReNeRo MeGaPacK	N/A	460	3.3%
Mute Sims X2 WIP	2018	12	3.0%
r2112	2007	60	2.7%
Technical Showcase 3	2018	196	2.7%
FoxyMix 4 - Nuclear Overdrive	≤2012	67	2.6%
Technical Showcase 2	2017	80	2.6%
Gensokyo Midnight	≤2012	77	2.5%

Table 3. Song packs densest in bracket jumps.

Pack name	Year	#charts	BJJ%
Feelin' Rusty 3	2018	51	50.2%
Squeaky Beds and Leaky Faucets	2018	125	46.7%
TYLR's Technical Difficulties	2018	33	44.9%
Technical Showcase 3	2018	196	41.7%
Mute Sims X2 WIP	2018	12	40.9%
Jimmy Jawns 2	2015	55	39.7%
Technical Showcase 2	2017	80	38.2%
Keyboard Collaboration III	≤2012	17	38.1%
Chicago Timing Authority	2018	33	36.3%
DVogan's Tech Support 2	2018	32	36.1%

Table 4. Song packs with the most jumps bracketable.

Author name	#charts	BJS%
Paul J Kim	45	9.9
sssmsm	45	4.4
Liam	11	3.4
Snooze	22	2.5
M. Emirzian	26	2.2
bblum	40	1.9
Ninevolt	10	1.7
B. Vergara	95	1.6
C. Emirzian	18	1.6
B. Dinh	10	1.6
Renard	51	1.5

Table 5. Step cartographers who write charts densest in bracket-jumps.

Author name	#charts	BJJ%
Paul J Kim	45	71.8
Rust	97	51.7
bblum	40	49.3
Liam	11	44.5
Snooze	22	38.1
Rems	10	36.5
sssmsm	45	32.9
Loak	19	31.7
Little Matt	42	31.4
Paparazzi	10	29.6

Table 6. Step cartographers whose jumps are most bracketable.

pack collection website, facebook group, or laptop filesystem metadata could accurately date enough packs before then to give meaningful sample sizes for each year. I also decided to exclude tournament packs such as ECFA from this analysis, which curate existing stepcharts written possibly long ago.

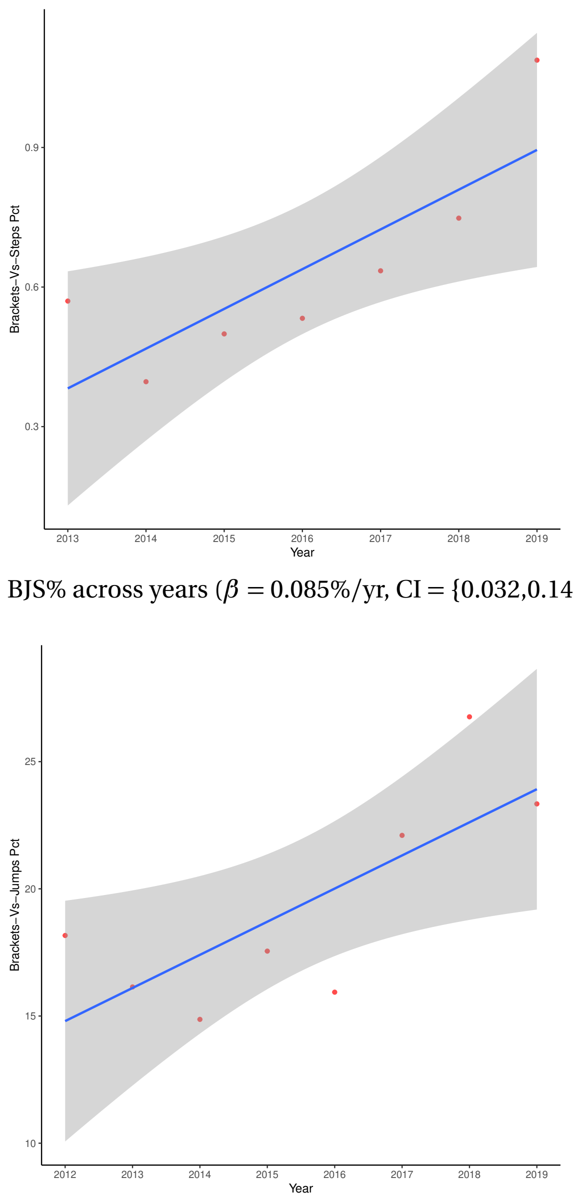
Statistical significance. Stepcharts thus dated, I then summed the total steps, jumps, and brackets published in each year, and analyzed the resulting overall BJS% and BJJ% as a linear regression. Figure 6 shows the best-fit and 95% confidence intervals, plotted in R (R Core Team 2018) with ggplot (Wickham 2016). For BJS%, the ≤2012 bucket actually produced the highest data point (at 11.8%); I suspect this simply represents the fact that jumps were more popular overall during ITG's nascency (as corroborated by Table 3). Owing to this and also to its nature as an aggregation over nearly a decade (which spoils the linear model anyway), I chose to exclude it from BJS%, but kept it for BJJ%, which overall jump count has no bearing on. (Its linear fit has $\beta = 0.012\%$, CI = {-0.08,+0.10}, not significant.) For the two pictured distributions, their confidence intervals do not include 0, i.e., a flat line representing no growth, so I conclude bracket jumps' increased popularity is statistically significant.

Bias. This analysis is prone to selection bias in my own pack downloading habits: supposing I suddenly became more interested in playing brackety charts recently, that would certainly influence how many bracket-jumps appeared each year in this corpus. However, I believe the BJJ% measurement adequately compensates for this, for if I suddenly became interested in stamina instead of technical and added a bunch of 10-minute trance packs, that would impact only BJS%. As far as I know, I show no preference for more or less jumpy charts overall than the community average, leading me to trust the BJJ% test.

5. Discussion

I confirmed in stepmania the bracketiness of each of Section 4.1's high scorers, and happily observed no false positives (i.e., all charts seemed to intend the player to bracket), surprising myself for the 3rd time running at the accuracy of the algorithm. I did observe some false negatives, i.e., jumps seeming intended to be bracketed that the algorithm wouldn't catch. In Beach Party (Nero 2017), shown in Figure 7(a), the "mine jump" forces the player to reset her footing, removing the right foot's initial presence on U to allow LU to be bracketed. This would actually be pretty easy to fix (just have the algorithm parse mines like, at all) but it's the day of the deadline and I already ran the experiment, so. Another class of false positives showed up in koopa bling (Ali 2018) (Figure 7(b)), where

Their full results are posted at: `https://tinyurl.com/bracketiest`



(a) BJS% across years ($\beta = 0.085\%/\mathrm{yr}$, $\mathrm{CI} = \{0.032, 0.14\}$).

(a) BJJ% across years ($\beta = 1.30\%/\mathrm{yr}$, $\mathrm{CI} = \{0.40, 2.21\}$).

Figure 6. Linear best fit of bracket jumps' popularity across the years of ITG's history. Grey area depicts 95% CI.

one foot is fixed to a hold, affecting the footing of subsequent jumps.[2] For kicks, I also show the source of the "I'm sorry" stepchart description in Figure 7(c), from Zombie Sunset (Sorry 2016) (Table 2); note here the DUR bracket-triple-jumps in the third measure, which the algorithm currently cannot process.

Many old charts written without regard to jump patterning allow for many of their jumps to be bracketed anyway. I noticed an old stepchart for One-Winged Angel (植松伸夫 1997) ranking high in the list of overall total bracket-jumps, with 182, simply because it's long and has a lot of steps, but upon playing, the patterns definitely do not feel intentional, and it even includes a section of stream with wholly unbracketable jumps interspersed. Figure 7(d) shows this section; in this stream, the player would expect to step the red and blue arrows with her right foot and the lime ones with her left, but note how the jumps (hint: all in blue arrows, unless you're reading this on dead trees) occur as a mix of left and right arrows, with even a UD "candle" jump making an appearance. However, it measures an unremarkable 9 BJS% and 38 BJJ%—I just wanted to feature it as an example of what non-bracketable jump stream looks like.

Authors seem to realize when their charts are too bracket jumpy for comfort: several of the charts appearing in the highest ranks of the spreadsheet had their step author field filled in as "Stupid" or "I'm sorry". In fact, I personally found the charts with 100 BJJ%, i.e., *all* jumps appearing therein were bracketable, to be more tasteful than those with more total BJS% but a few normal jumps as well. I'd hypothesize this is because to achieve 100 BJJ% requires a certain intentionality, resulting in better chart design overall (note that such charts necessarily include no LR or DU jumps whatsoever). Either that, or the more uncomfortable ones include certain extremely turny/candley/doublesteppy patterns around their brackets that causes the algorithm to count them as normal jumps instead. Anyway, Figure 7(e) shows the chart with most bracket-jumps (35) among ones with 100 BJJ%, which I tried out for myself and found quite enjoyable (ATB 2018). Amusingly, Encore (Reen 2017) features a Hard chart in addition to its Challenge, which former ranks second place to Electrical Paradise among 100 BJJ%s with 29 brackets. Upon inspection, I found this chart identical to the Challenge (upcoming in Figure 9(a)), only the crossover brackets (to be discussed later in Section 6.2) having been replaced by easier, normal ones.

6. Never Work

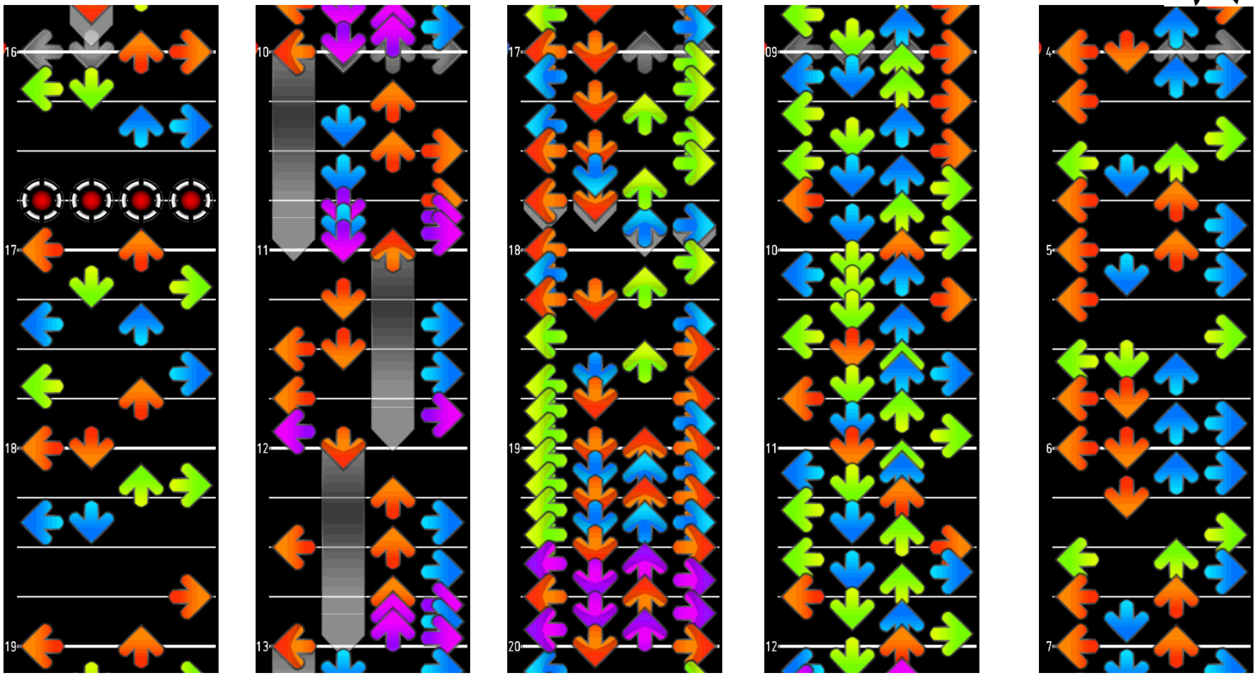
The only thing that impresses the research community more than overdelivering on your future work promises is to overdeliver twice; hence, I shamelessly reuse the joke of this section name from last time.

6.1 UI

It would be cool to integrate this (and the preceding two algorithms (Blum 2016, 2017)) into the community's prevailing stepmania theme. Currently, as shown in Figure 8(a), the song preview screen wastes considerable space displaying the count of irrelevant chart aspects such as mines (now used primarily for signaling the pres-

[2] The algorithm actually foots them correctly in this chart by sheer luck (an even number of preceding doublesteps maintains footing parity), but these would be harder to handle in general.

Their full results are posted at: `https://tinyurl.com/bracketiest`



(a) Beach Party, 15 (Nero 2017)

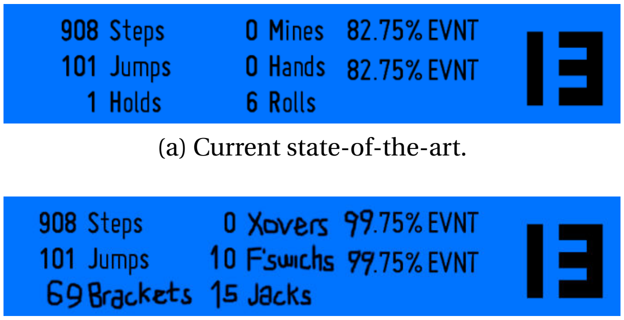
(b) koopa bling, 69 (Ali 2018)

(c) Zombie Sunset, 17 (Sorry 2016)

(d) 1-Winged Angel, 14 (植松伸夫 1997)

(e) Electrical Paradise, 12 (ATB 2018)

Figure 7. Example real-world charts with (a) false negative jumps that mines render bracketable after all, (b) jumps whose footing is affected by preceding holds fixing one foot in place, (c) brackets the chart author was sorry about, (d) "retro" style unbracketable jump-stream, and (e) intentionally many true bracket jumps from the BJJ% category winner.



```
 908 Steps       0 Mines   82.75% EVNT
 101 Jumps       0 Hands   82.75% EVNT
   1 Holds       6 Rolls
                                         13
```

(a) Current state-of-the-art.

```
 908 Steps       0 Xovers   99.75% EVNT
 101 Jumps      10 F'swichs 99.75% EVNT
  69 Brackets   15 Jacks
                                         13
```

(b) Artist's conception.

Figure 8. Song info preview panel for a recent bracket jump-heavy stepchart (Sanchez 2018).

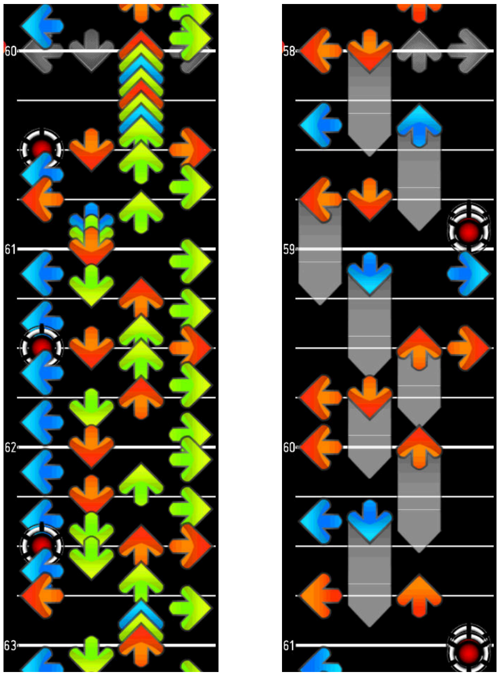
ence of footswitches) and hands (now mostly stepped with the feet by bracketing anyways). I bet the community might actually pay attention to my work—the holy grail of research, honestly—if popular themes used it to show precisely how technical a chart was, as in (b).

6.2 Crossover Brackets

Whereas with normal crossovers (e.g., in the sequence L-U-R, stepping the R with the left foot), the player's foot can pretty much point whichever way she finds most comfortable; however, crossover brackets (e.g., in the sequence L-U-DR...), the player's foot must point backwards (...stepping the DR's R with her left heel and D with toes), inducing extra turniness.[3] Such jumps could instead just be stepped with both feet as normal, possibly inducing doublesteps or jacks, and indeed that is how the algorithm presently handles them.

However, ambitious stepchart authors have recently experimented with encouraging the player to bracket-jumps while crossed-over. Such charts attempt to force, or at least hint, these jumps to be bracketed via additional stepchart elements: in Figure 9(a), mines on the left arrow force the player to remove her left foot in preparation for the crossover (and being in the middle of fast stream, further discourages doublestepping), and in Figure 9(b), extending one of the arrows as a hold encourages the player to bracket the subsequent jump with the other foot. In (b)'s case, the player must also switch feet on each of the non-hold arrows, effectively making these crossover-footswitch-brackets, which would truly be a wonder to identify programmatically. Note however the high measure counter in both, meaning these occur quite late in

[3] The other way, e.g. stepping the D with the left heel and R with toes, while your right foot is fixed on U, is extremely uncomfortable at speed. Trust me on this one.



(a) Encore, 12 (Reen 2017)

(a) Atariwave, 10 (Tuuc 2018)

Figure 9. Crossover bracket-jumps must be signaled via either (a) mines or (b) holds to remove ambiguity.

the song; both charts first spend some time "teaching" the player both to expect bracket jumps and how it intends to signal crossovers in general—so if a human is not expected to understand them at first glance, I think it is fair to leave out of the algorithm too for now.

6.3 Footswitch Brackets

I put this section here only just so I could have something to forward-reference back in Section 3, but not to actually write anything. I always wanted to do that, you know? I mean, footswitch brackets are a real thing, but still. ¯_(ツ)_/¯

6.4 Obligatory Machine Learning Section

I guess you could skip all this fiddly "algorithm" stuff by hopping on the pads yourself for a few rounds, playing a variety of technical charts, and just training a neural network based on how you stepped the patterns. It would probably even learn to fake crossovers more at the ends of songs than at the beginnings, where you're more likely to be physically tired. And like, do you really want that kind of bias in your dataset?

Remember kids, friends don't let friends use ML for problems that are more fun to solve by hand.

7. Conclusion

Bracket jumps is, in fact, statistically significantly, the new crossovers or something.

Acknowledgments

neuropantser (Michael T. Lawson) provided invaluable eleventh-hour (literally) advice on Section 4.2's statistical analysis, up to and including making the graphs for me. Greg Hanneman unknowingly helped copy-edit my hyphenation. And much love to the SIGBOVIK community.

References

m. Ali. koopa bling. UPS 4, 2018.

ATB. Electrical paradise. Chicago Timing Authority, 2018.

B. Blum. Which ITG stepcharts are turniest? SIGBOVIK, 2016.

B. Blum. A boring follow-up paper to "Which ITG stepcharts are turniest?" titled, "Which ITG stepcharts are crossoveriest and/or footswitchiest?". SIGBOVIK, 2017.

B. Blum. Practical concurrency testing, or: how I learned to stop worrying and love the exponential explosion. Ph.D. dissertation CMU-CS-18-128. Carnegie Mellon University, 2018.

w. dril. using my turn at a karaoke bar to try to do a 5 miunte routine that basically just says bryan singer is the new gawker writer or something. Twitter, 2019.

J. Nero. Beach party. Technical Showcase 2, 2017.

R Core Team. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria, 2018.

S. Reen. Encore. Technical Showcase, 2017.

S. Sanchez. Divine. Technical Showcase 3, 2018.

I. Sorry. Zombie sunset. Jimmy Jawns 2, 2016.

Tuuc. Atariwave. Technical Showcase 3, 2018.

H. Wickham. ggplot2: Elegant graphics for data analysis. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4.

植松伸夫. One-winged angel. Final Fantasy VII (stepchart pack and date unknown), 1997.

The Computational Theory of Lord Voldemort's Dark Magic

Huilian Sophie Qiu
Carnegie Mellon University
hsqq@cmu.edu

Hui Yang
Carnegie Mellon University
yanghui@cmu.edu

***Abstract*—For a long time, we do not have enough understandings of the magic happening around us. We accidentally mention He-Who-Must-Not-Be-Named and then, we got in trouble. Double, double, toil and trouble. And you better watch out and hide in a hole, because he'll reach down your throat and swallow your soul... Voldemort is coming to town!**

We wish to educate our non-magical friends, Muggles, the theory behind the magic. We found that Muggles' computing technologies can be used to simulate and explain magic. Fire burns and cauldron bubbles. Ladies and gentlemen, fellow witches and wizards, we now proudly introduce our new interdisciplinary area of research – Computational Magic Theory!

I. Introduction

While the novels and movies of *Harry Potter* have become the top sellers, the images of witches and wizards living a middle age life style have been deeply planted into the Muggles, *i.e.*, non-magical folks. However, as we entered the twenty-first century, wizards and witches have been seeking inspirations from Muggle technologies to improve their life quality. On the other hand, seeing that Muggles' attitude towards magic has become more positive, many wizards and witches seek to foster conversations and communications between wizarding members and Muggles.

To eliminate the stereotype and increase Muggles' understandings of the wizarding world, we propose and define **Computational Magic Theory** which explores how to use Muggle's computing technology to emulate witchcraft and wizardry. It is a new crosscutting research area that allows witches and wizards to explain magic to Muggles as well as Muggles to understand witchcraft. With the Green Computing concept taken into concern, when seeking computing techniques to explain certain magic, we choose the simplest yet most effective computing tools over those more powerful but much more complex and energy consuming ones. By energy, we include, of course, one's brain energy that puts into learninng such technique.

In this paper, we present a comprehensive case study to demonstrate the methodology of **Computational Magic Theory**. In our experiment, we firstly divided the main question into small ones by *Diffindo* using a 13½" long phoenix core yew wand, a replica of Lord Voldemort's wand, since the main question is about him. Then we applied computational models that best fit the scenarios and tested the models with data gathered from film clips. Results show that our models are robust, and the **Theory** is credible. In the end, we *Obliviated* Mundungus Fletcher who was hanging around with an Extendable Ear and caught by us when we were doing the experiment, so that he would not sell our findings for thirty-seven Galleons, fifteen Sickles, and three Knuts to someone else before we finish this paper (otherwise you probably would not be the first one reading about this great Theory)..

II. Description of Magic

The magic for which we are aiming to find a computational theory is the way Lord Voldemort put a jinx on his name in the seventh *Harry Potter* novel *Harry Potter and the Deathly Hallow*. Saying Voldemort's name "breaks protective enchantments, it causes some kind of magical disturbance" [1]. In the book, Ron Weasley did not explain how the Taboo worked. Nevertheless, we can surmise that it must be some kind of dark magic. The novels never explained any theories behind any dark magic, which understandable because the books were written from the good wizards' perspective and they were not supposed to be experts on dark magics.

However, the same magical effects can be achieved by different mechanisms, some are dark and some are white. Therefore, although we intend to discuss Voldemort's name Taboo in this paper, we are only using it as an example of a certain type of magic effect that allows people to locate the person who speaks certain kinds of words. In fact, this magic belongs to a broader kind of magic that can track people, including the location of underage wizards performing magic, which is beyond the scope of this paper.

III. Computational Theory for Name Taboo

The theory we propose involves three parts. The input of the model is a mixed of all people's conversation. The output is the location of the people who spoke the word "Voldemort." The first part of the model is an Independent Component Analysis (ICA) that can separate the sources of speeches. The second part is a Hidden Markov Model (HMM) that can identify certain words in the speech. The third part is a system that incorporates information from at least three computing nodes to locate the person who speaks the target word. To implement the third part, we used Hagrid's pet, Fluffy [2], shown in Figure 1, because it can use its three heads to locate the object in front of it. Figure 2 shows the entire flow of the system.

Fig. 1: Hagrid’s dog, Fluffy

Fig. 3: The merged audio wave looks like an obscurus.

IV. Experiment

The data used in this study were audio clips taken from the fifth *Harry Potter* movie, *Harry Potter and the Order of Phoenix* [3]. We took one sentence that mentioned “Voldemoret” or “You-Know-Who” from each of the top ten characters, who have the most number of lines in the movie. For each of the top ten characters, we also included one sentence that is irrelevant to the Dark Lord. We set up ten speakers and microphones, one for each character, in a Wean Hall office. Many of the Wean Hall offices are completely shut out from the outside world because they do not have windows or only have windows facing a corridor. The gloom in those offices suit the dark theme of our paper. At the same time, all ten speakers play their own character’s conversation and ten microphones capture the sound.

The merged audio wave captured by one of the microphones is shown in 3. Its shape looks very similar to an obscurus [4].

V. Results

Because we brought our computers too close to some magical buildings, we lost all our data and results. But no matter, we managed to record the shape of the waves and reproduced it here. To eliminate the subjectivity, we had two people, i.e., both of our authors, to recall the shape independently. Then we met and compared and take the intersection of what we remembered. To maximize the inaccuracy, this process was done while the two authors were drinking butter beer at the Three Broomsticks. As a result, the sound waves were written on napkins.

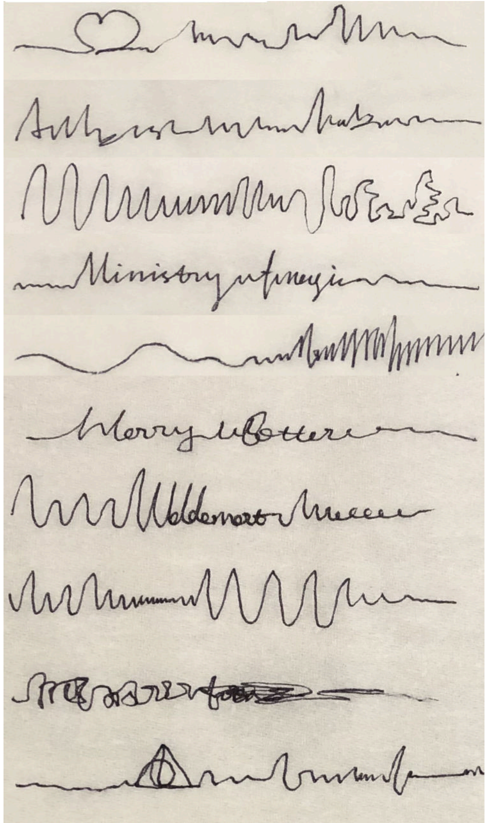


Fig. 4: ICA results

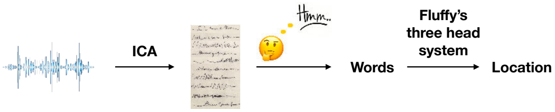


Fig. 2: The computational theory for name taboo.

Figure 4 shows the separated sound waves captured by one of the microphones. We can see very clearly that some of the sound waves already resemble some of the words.

Using fluffy’s heads, we were able to recover the location of each speaker. The only drawback of this process is that your head might be bitten off.

VI. Discussion

Through our experiment, we showed the spirit and integrity of our proposed computational theory. The ICA model we built can successfully distinguish the voice from the person who broke the Name Taboo, and the HMM model can provide estimations of distance and location with satisfying accuracy. Finally, Fluffy helps us locate the speaker.

While our case demonstrated how we used the **Theory** to explain the Dark Lord's dark magic, we did not practice on "white" magic such as Protean Charm, which may be subject to future work.

In the long run, after trying out different cases, we could research on the tradeoffs between computational models for witchcraft and certain domains where magical societal issues might arise, such as dark magic vs. white magic, Society for the Promotion of Elfish Welfare aka SPEW, and the abuse of Felix Felicis (if there really is such a thing). These would lead us to a new research field called **Societal Computing in Magical Context** which derives from the Societal Computing in the muggle world.

References

[1] J. K. Rowling, *Harry Potter and the Deathly Hallows*. Bloomsbury Publishing, 2007.

[2] H. P. Wiki, *Fluffy snarling at his discoverers on this day in 1991*. Harry Potter Wiki, 2001. [Online]. Available: https://harrypotter.fandom.com/wiki/13_September

[3] D. Hayman and D. Barron, *Harry Potter and the Order of the Phoenix*. Warner Bros. Entertainment Inc., 2007.

[4] H. P. Wiki, *Obscurus*. Harry Potter Wiki, 2016. [Online]. Available: https://harrypotter.fandom.com/wiki/Obscurus

All You Need is Dogball

Kai Arulkumaran
Imperial College London
London, UK
ka709@ic.ac.uk

Matthew Kelcey
Victoria, Australia
matthew.kelcey@gmail.com

Andrew Brock
Heriot-Watt University
Edinburgh, UK
ajb5@hw.ac.uk

Abstract

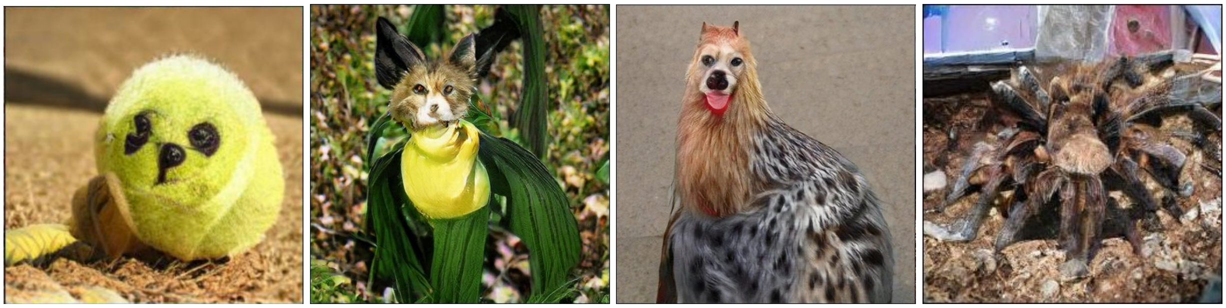
The year is 2019, and humanity is on the brink of destruction. The latter fact has nothing to do with the current state of AI, but if we do manage to survive the next few decades, it may well do. If AI can now beat us at our own simulated war games, or convince us that there exists a secret herd of unicorns in South America, what next? All things considered, we propose that a reasonable option is to always look on the bright side of life. More specifically, we chronicle here the conception of the well-loved AI creation, Dogball, and its later adventures on the interwebs.

1 Introduction

First, we were told that attention is all you need (Vaswani et al., 2017). Then, we were told that, just maybe, you *didn't* need attention (Press & Smith, 2018). And somewhere along the way, we were also told that all we needed was CNNs (Chen & Wu, 2017), but by that point Bored Yann LeCun was getting a little repetitive and we kind of ignored that one #torched. So now, we're here to say that all you need is Dogball, because YOLO (Redmon & Farhadi, 2018).

The origins of Dogball lie in the Inception wars of 2016-2018 (Salimans et al., 2016), in which research groups worldwide were competing to make the prettiest, most high resolution faces in the name of science. Despite heroic efforts to reduce GAN violence (Albanie et al., 2017), the arms race escalated in recent years, culminating in the notorious BigGAN[1] (Brock et al., 2019). With one fell swoop and a lot of TPUs (Buchlovsky et al., 2019), BigGAN blew other GANs out the water, putting an end to the conflict[2].

Many experiments went into the creation of the final BigGAN models, and these were duly chronicled in the appendix. Indeed, the community noted the level of detail available, a feat usually reserved for works by Hochreiter (Klambauer et al., 2017). Experiments ranged over hyperparameters, regularisation strategies, noise distributions[3], and much more. However, the most serendipitous finding was Dogball, a creation from a BigGAN in the middle of training. Dogball and his family of chimeras (see Figure 1) were the result of a phenomenon that was named *class leakage*, bringing literal meaning to the maxim that deep learning is alchemy.



(a) Dogball (b) Catflower (c) Hendog (d) Nope

Figure 1: Dogball family portraits. (a-c) Dogball, Catflower and Hendog are all members of the *classus leakus* family. (d) Nope is extended family from the father's side.

[1] Whose aliases also include "the BFG" (big feedforward GAN).

[2] At least until StyleGAN showed up a few months later (Karras et al., 2018).

[3] RIP Bernie the Bernoulli BigGAN, your hypercubic binary latent space was beautiful, but alas you were not amenable to the truncation trick.

2 Cultural Impact

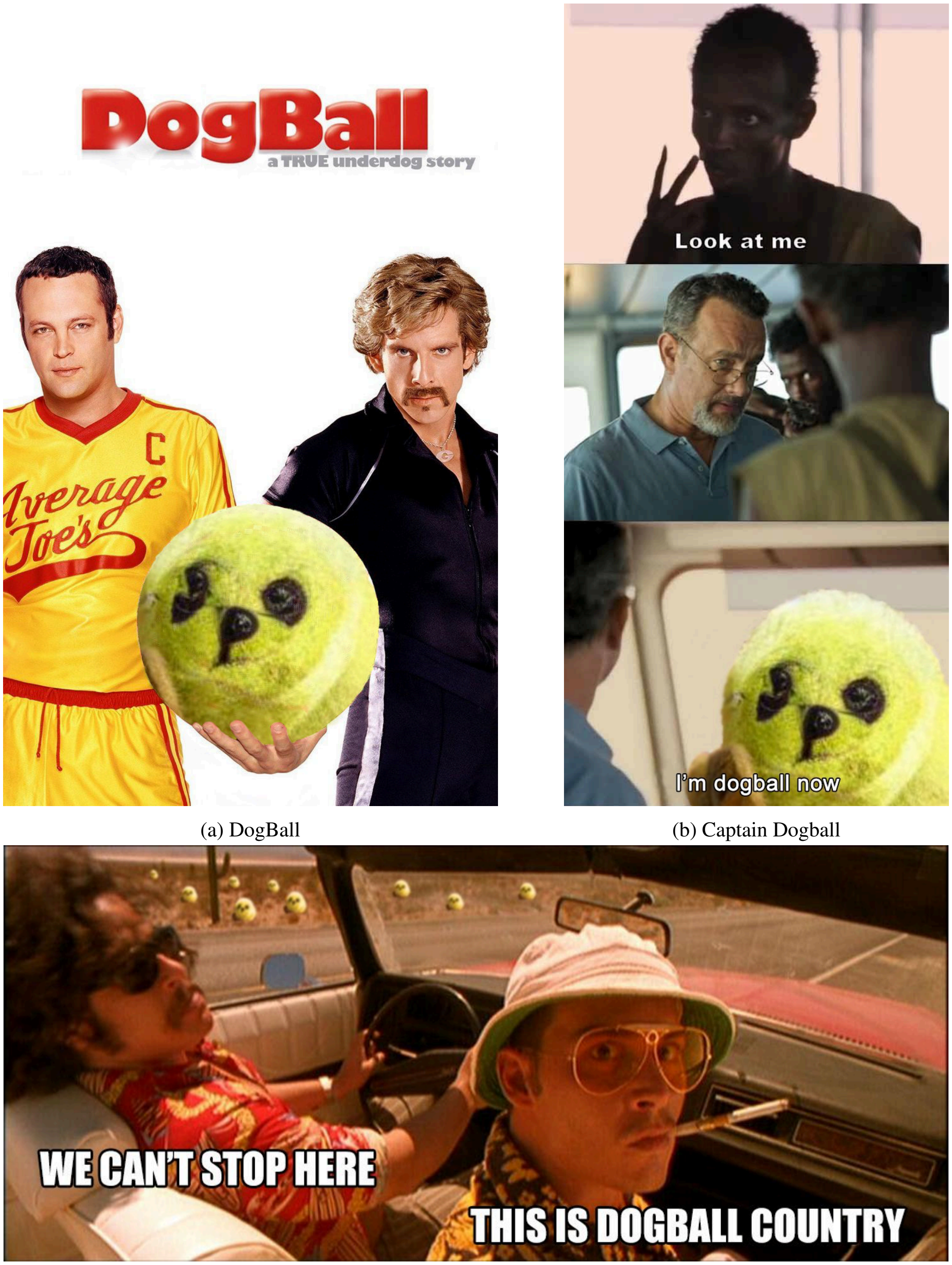
Deep learning research is no stranger to whimsy. From figuratively (Kaiser et al., 2017; Schmidhuber, 2018) to literally outrageous (Shazeer et al., 2017) names, deep learning researchers are fond of their wordplay (Donahue et al., 2017; Tomczak & Welling, 2018). The community is also a fan of animals, with a veritable zoo of models, including MAMLs (Finn et al., 2017), Reptiles (Nichol & Schulman, 2018), SNAILs (Mishra et al., 2018) and even DRAGANs (Kodali et al., 2017). Given all of this, it was perhaps inevitable that we could put all the seriousness aside for a moment[4], and relish in the glory that was Dogball (see Figure 2).

Figure 2: GANs are only useful for making pretty pict-ALL GLORY TO THE DOGBALL!

Dogball appealed through classic memes (see Figure 3) and other pop culture references (see Figure 4). Despite the small backlash to the proliferation of Dogball memes (see Figure 5), resistance was futile (see Figure 6), and was eventually assimilated (see Figure 7).

Figure 3: On the phone with the English-speaking South American unicorns, another example of AI-created phenomenon (Radford et al., 2019).

[4] Current topics included how many meta-'s to include in your meta-learning algorithm, and choosing which Sesame Street character to name your new NLP model after.



(a) DogBall

(b) Captain Dogball

(c) Fear and Dogball in Las Vegas

Figure 4: Films are parables for modern times. (a) is the source of the common adage, “If you can dodge a wrench, you can dodge a Dogball.” (b) is about dressing for the job you want. (c) drugs are bad, OK?

STOP TRYING TO MAKE DOGBALL MEMES HAPPEN
IT'S NOT GOING TO HAPPEN!

Figure 5: Oh my God, Mat! You can't just ask people to stop making dogball memes!

Figure 6: "Likelihood-based models have no chance to survive make your time."

Figure 7: Honestly, neither do we.

3 Conclusion

Despite the short-lived nature of fame on the internet and the even shorter-lived nature of state-of-the-art results in deep learning, the legacy of BigGAN and Dogball lives on in various places, such as in the custom emoji of various research lab Slack channels. The authors hope that this work serves as a reminder that, once in a while, it's nice to instead work on the frivolous uses of AI.

References

Samuel Albanie, Sébastien Ehrhardt, and João F Henriques. Stopping GAN Violence: Generative Unadversarial Networks. In *SIGBOVIK*, 2017.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. In *ICLR*, 2019.

Peter Buchlovsky, David Budden, Dominik Grewe, Chris Jones, John Aslanides, Frederic Besse, Andy Brock, Aidan Clark, Sergio Gómez Colmenarejo, Aedan Pope, et al. TF-Replicator: Distributed Machine Learning for Researchers. *arXiv preprint arXiv:1902.00465*, 2019.

Qiming Chen and Ren Wu. CNN Is All You Need. *arXiv preprint arXiv:1712.09662*, 2017.

Chris Donahue, Zachary C Lipton, and Julian McAuley. Dance Dance Convolution. In *ICML*, pp. 1039–1048, 2017.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic Meta-learning for Fast Adaptation of Deep Networks. In *ICML*, 2017.

Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One Model to Learn Them All. *arXiv preprint arXiv:1706.05137*, 2017.

Tero Karras, Samuli Laine, and Timo Aila. A Style-based Generator Architecture for Generative Adversarial Networks. *arXiv preprint arXiv:1812.04948*, 2018.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing Neural Networks. In *NIPS*, pp. 971–980, 2017.

Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. On Convergence and Stability of GANs. *arXiv preprint arXiv:1705.07215*, 2017.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A Simple Neural Attentive Meta-learner. In *ICLR*, 2018.

Alex Nichol and John Schulman. Reptile: A Scalable Metalearning Algorithm. *arXiv preprint arXiv:1803.02999*, 2018.

Ofir Press and Noah A Smith. You May Not Need Attention. *arXiv preprint arXiv:1810.13409*, 2018.

Alex Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. Technical report, OpenAI, 2019.

Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767*, 2018.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs. In *NIPS*, pp. 2234–2242, 2016.

Juergen Schmidhuber. One Big Net For Everything. *arXiv preprint arXiv:1802.08864*, 2018.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-gated Mixture-of-experts Layer. In *ICLR*, 2017.

Jakub Tomczak and Max Welling. VAE with a VampPrior. In *AISTATS*, pp. 1214–1223, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *NIPS*, pp. 5998–6008, 2017.

ON THE TIME COMPLEXITY OF THE VERIFICATION OF THE FACTORIZATION OF $2^{67} - 1$

Isaac Grosof
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
igrosof@cmu.edu

Isaac Grosof
Computing Hardware

March 13, 2019

ABSTRACT

In 1903, Frank Nelson Cole [4] demonstrated that the 67th Mersenne number, 147573952589676412927, is equal to $193707721 \times 761838257287$, disproving Marin Mersenne's claim of primality from 1644 [9]. Cole demonstrated this equality by silently multiplying the two factors on a blackboard, receiving a standing ovation [3, 5]. Modern sources include the additional embellishment that this demonstration took one hour. We find this claim suspicious, as we grow bored with hour-long lectures in the best of circumstances.

To resolve this discrepancy, we investigate the time complexity of the multiplication of middling-large numbers. We use similar computational hardware to the original demonstration, namely the second author. We investigate a variety of multiplication algorithms, including algorithms that Cole might have used and more recent algorithms. We find that Cole could have performed the demonstration in about 10 minutes.

1 Introduction

In 1644 [9], Marin Mersenne claimed without proof that $2^{67} - 1$ is prime. In 1903 [4], Frank Nelson Cole disproved with claim by exhibiting the two prime factors of $2^{67} - 1$: 193707721 and 761838257287. An account of this demonstration is given by N. T. Gridgeman [5]:

> At a mathematical meeting in New York in 1903, F. N. Cole walked on to the platform and, without saying a single word, wrote two large numbers on the blackboard. He multiplied them out in longhand, and equated the result to $2^{67} - 1$. (Subsequently, in private, Cole said that those **few minutes** at the blackboard had cost him three years of Sundays.)

Wikipedia [1], whose accuracy on historical figures has been found lacking [6], gives the following account of the same event, citing N. T. Gridgeman's account above as its only relevant source:

> On October 31, 1903, Cole famously made a presentation to a meeting of the American Mathematical Society where he identified the factors of the Mersenne number $2^{67} - 1$, or M_{67}. ... During Cole's so called "lecture", he approached the chalkboard and in complete silence proceeded to calculate the value of M_{67}, with the result being $147,573,952,589,676,412,927$. Cole then moved to the other side of the board and wrote $193,707,721 \times 761,838,257,287$, and worked through the tedious calculations by hand. Upon completing the multiplication and demonstrating that the result equaled M67, Cole returned to his seat, not having uttered a word during the **hour-long** presentation. His audience greeted the presentation with a standing ovation. Cole later admitted that finding the factors had taken "three years of Sundays."

Note the key discrepancy: a “few minutes" at the blackboard became an “hour-long" lecture. Given that both accounts agree that the silent demonstration received a standing ovation, the former is far more plausible.

Another modern account [2] also describes the demonstration as taking “nearly an hour", while citing a source [3] that makes no such claim.

To resolve this discrepancy in the literature, we turn to simulation. We investigate the time complexity of the multiplication $193707721 \times 761838257287$ under various algorithms. We use the second author as computing hardware, which is of approximately the same computational performance as that used in the original demonstration.

We investigate four multiplication algorithms, each performed by hand on paper in base 10:

- Lattice multiplication
- Double-halve multiplication
- Quarter-Square multiplication
- Karatsuba multiplication

In addition to recording the runtime of each algorithm, we also investigate the number of digits written during the computation, which we observe to be highly correlated with runtime. We also comment on the fault-tolerance of each algorithm, which we find to be a major source of variance in runtime.

We find that the fastest algorithm, lattice multiplication, consistently takes 10 to 12 minutes. We would be far more likely to applaud 10 minutes of silent multiplication than an hour thereof, so we judge the older accounts [3, 5] to be the true ones.

2 Lattice Multiplication

In the lattice multiplication algorithm, the two multiplicands are written at the top and right of a grid. The product of each pair of digits is written in the cell at the intersection of the row and column, with the high digit to the upper left and the low digit to the lower right. Diagonals are summed, then those sums are summed to give the final result.

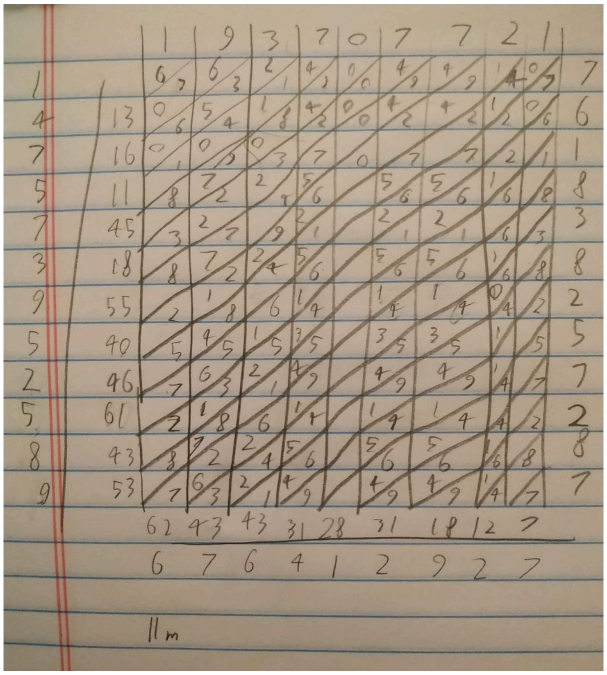


Figure 1: Lattice multiplication computation

Figure 1 shows the computation. Note that partway through the computation, the computing hardware stopped writing zeros, because it couldn’t be bothered.

This method involved writing down 249 digits, which took 11 minutes, resulting in a computation frequency of 0.38 Hz. For multiplication of an m digit number by an n digit number, approximately

$$2mn + 4(m + n)$$

digits need to be written down.

On fault-tolerance, this algorithm performs well: the computing hardware only made 3 errors, each of which affected one or two output bits. The computing hardware found each error found relatively easily, only costing a minute or two in total.

In the end, this algorithm took 11 minutes to compute, which was a bit boring but not too bad. We could give a standing ovation to this.

2.1 Long Multiplication

Long multiplication is just a worse version of lattice multiplication. It’s like lattice multiplication, but with lots of extra additions interspersed among the multiplications. It’s called "long" for a reason. It’s really a shame that long multiplication is the standard method taught to most people. This is nothing short of a major failing of the education system.

3 Double-Halve Multiplication

In double-halve multiplication, the smaller multiplicand is repeatedly halved while the larger is repeatedly doubled. Each pair where the halving of the smaller multiplicand produces an even number is crossed out. The remaining doublings of the larger multiplicand are summed, giving the product.

This algorithm essentially consists of a binary decomposition of the smaller multiplicand, which is used to perform a shift-and-add multiplication on the larger multiplicand.

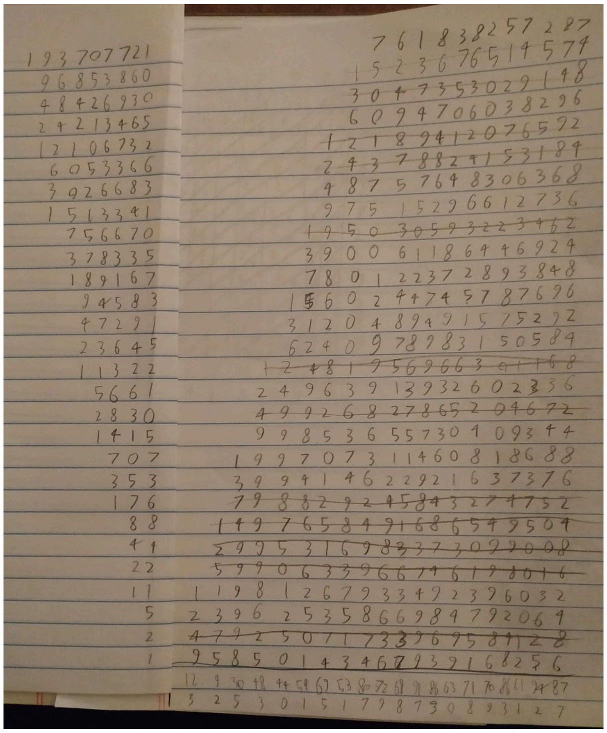
Figure 2 shows the computation. Note that only the first digit and the last two digits of the output are correct, because the computing hardware made a mistake partway through, and definitely isn't being paid enough to do this again.

This method required writing down 651 numbers, which took 21 minutes, resulting in a computation frequency of 0.52 Hz. For a multiplication of a m digit number by an n digit number, where $m \leq n$, approximately

$$(4 + m \log_2 10)(m + n)$$

digits need to be written down.

Based on the number of digits written as a proxy for runtime, we should expect double-halve multiplication to have better performance than lattice multiplication when $m \leq 0.423n$. Or it would if it wasn't for all of the errors.



193707721	761838257287
96853860	~~1523676514574~~
48426930	~~3047353029148~~
24213465	6094706038296
12106732	~~12189412076592~~
6053366	~~24378824153184~~
3026683	48757648306368
1513341	97515296612736
756670	~~195030593223462~~
378335	390061186446924
189167	780122372893848
94583	1560244745787696
47291	3120489491575292
23645	6240978983150584
11322	~~12481956966301168~~
5661	24963913932602336
2830	~~49926827865204672~~
1415	99853655730409344
707	199707311460818688
353	399414622921637376
176	~~798829245843274752~~
88	~~1597658491686549504~~
44	~~2995316983373099008~~
22	~~5990633966746198016~~
11	11981267933492396032
5	23962535866984792064
2	~~47925071733969584128~~
1	95850143467939168256
	12 9 30 48 44 54 67 53 80 72 68 91 86 63 71 70 86 61 94 87
	1 3 2 5 3 0 1 5 1 7 7 9 8 7 3 0 8 9 3 1 2 7

Figure 2: Double-halve multiplication computation

On fault-tolerance, this algorithm is terrible. A single error partway through is almost impossible to find and rectify, as demonstrated in Figure 2, beats us where.

In the end, this algorithm took 21 minutes to utterly fail to verify the factorization. Definitely no standing ovation.

4 Quarter-Square Multiplication

Quarter-square multiplication is based on the following identity:

$$xy = \frac{(x+y)^2}{4} - \frac{(x-y)^2}{4}$$

As a result, given a pre-computed table of $\frac{n^2}{4}$, one can efficiently perform multiplications. Of course, this is less "computation by writing on a blackboard" and more "computation by looking in a book", but the comparison is still interesting.

Given a quarter-square book with entries up to at least $193,707,721 + 761,838,257,287$, the demonstration could be performed with two lookups and a couple of additions and subtractions. However, that quarter-square book would have approximately 10^{12} words. At no more than 1000 words to the page and 10000 pages to the meter (see Figure 3), the book would stretch approximately 100 kilometers. As a result, the multiplication would have taken at least a couple of hours, given transportation methods available in 1903.

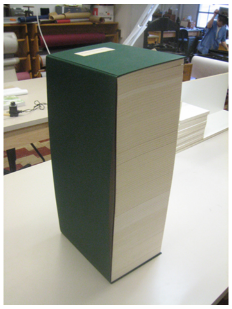


Figure 3: A 10,119 page book [10]. We're going to need a quarter-square table 100,000 times longer.

The largest quarter-square book ever published had entries up to $200,000$ [7], and was in print in 1903. Using this book to multiply the two numbers of interest would require breaking the multiplication into at least 6 sub-multiplications, requiring 12 lookups. Assuming a lookup takes in the range of 30 seconds to a minute, this method is competitive with the other multiplication algorithms in this paper.

With all that said, if someone silently looked up 12 numbers in a book and added them up, we would not give a standing ovation. We want to see the chalk fly!

5 Karatsuba Multiplication

Karatsuba multiplication was invented in 1960 [8], well after Prof. Cole's demonstration. Nonetheless, we may investigate whether it could have been used to improve the performance of the performance.

Karatsuba multiplication makes use of the following insight:

Let $x = x_1 b + x_0$ and let $y = y_1 b + y_0$. Then

$$xy = x_1 y_1 b^2 + (x_1 y_0 + y_1 x_0) b + x_0 y_0.$$

However, we can compute the middle term using the outer two terms and one additional multiplication:

$$x_1 y_0 + y_1 x_0 = x_1 y_1 + x_0 y_0 - (x_1 - x_0)(y_1 - y_0).$$

For the recursive calls, Karatsuba multiplication may be applied again, or another multiplication method may be used for the smaller multiplications.

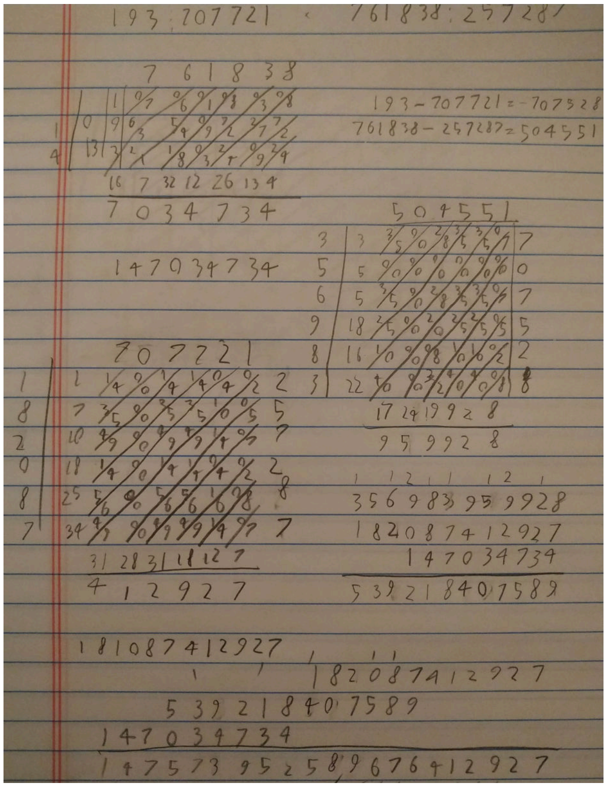


Figure 4: Karatsuba multiplication computation

Figure 4 shows the computation. Karatsuba multiplication was used with $b = 10^6$, and lattice multiplication was used for the recursive calls. Note that the computation hardware had a lot more fun calculating this way, it should do this more often.

This method required writing down 460 digits, which took 17 minutes, resulting in a computation speed of 0.45 Hz. For a multiplication of two n digit numbers, approximately

$$17n + \frac{3}{2}n^2$$

digits need to be written down.

Based on digits written as a proxy for runtime, we should expect Karatsuba multiplication to be an improvement over lattice multiplication when $n > 18$.

On fault tolerance, mistakes were reasonably locally recoverable, thanks to the use of lattice multiplication for the recursive calls. That being said, the use of many different types of operations did increase the error rate.

This algorithm took 17 minutes, but we got to do some fun and interesting computations along the way. Ovation awarded, but we'd probably stay seated.

6 Conclusion

Prof. Cole probably took about 10 minutes to show that

$$193,707,721 \times 761,838,257,287 = 147,573,952,589,676,412,927.$$

We don't know where you got an hour from, Wikipedia, but you're (probably) wrong. Also, he either used or should have used lattice multiplication to do it.

Other multiplication methods have their merits: Quarter-square multiplication might be faster, given a large book's worth of precomputation, while Karatsuba multiplication is a lot of fun. Double-halve multiplication doesn't have merits. Please don't make us do it again.

References

[1] Anonymous. Frank nelson cole. `https://en.wikipedia.org/wiki/Frank_Nelson_Cole`. Accessed: 2019-03-13.

[2] Anonymous. Lecture sans paroles: the factors of m67. `https://thatsmaths.com/2016/06/30/lecture-sans-paroles-the-factors-of-m67/`. Accessed: 2019-03-13.

[3] Eric T. Bell. *Mathematics, Queen and Servant of Science.* 1952.

[4] Frank N Cole. On the factoring of large numbers. *Bulletin of the American Mathematical Society*, 10(3):134–137, 1903.

[5] N. T. Gridgeman. The search for perfect numbers. *New Scientist*, 18(334):86–88, 1963.

[6] Lucy Holman Rector. Comparison of wikipedia and other encyclopedias for accuracy, breadth, and depth in historical articles. *Reference services review*, 36(1):7–22, 2008.

[7] Neville Holmes. Multiplying with quarter squares. *The Mathematical Gazette*, 87(509):296–299, 2003.

[8] Anatolii Alekseevich Karatsuba and Yu P Ofman. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk*, volume 145, pages 293–294. Russian Academy of Sciences, 1962.

[9] Marin Mersenne. *Cogitata Physica-Mathematica*. 1644.

[10] Bill Voss. 10,000 page book bound at conservation lab. `https://blog.lib.uiowa.edu/preservation/2011/01/07/10000-page-book-bound-at-conservation-lab/`. Accessed: 2019-03-13.