

THE ASSOCIATION FOR COMPUTATIONAL HERESY

PRESENTS

A RECORD OF THE PROCEEDINGS OF

SIGBOVIK 2015

The ninth annual intercalary robot dance party in celebration of workshop on symposium about 2⁶th birthdays: in particular, that of Harry Q. Bovik

Carnegie Mellon University

Pittsburgh, PA

April 1, 2015



Association for Computational Heresy

Advancing computing as Tomfoolery & Distraction

SIGBOVIK

A Record of the Proceedings of SIGBOVIK 2015

ISSN 2155-0166

April 1, 2015

Copyright is maintained by the individual authors, though obviously this all gets posted to the Internet and stuff, because it's 2015.

Permission to make digital or hard copies of portions of this work for personal use is granted; permission to make digital or hard copies of portions of this work for classroom use is also granted, but seems ill-advised. Abstracting with credit is permitted; abstracting with credit cards seems difficult.

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



SIGBOVIK 2015

Message from the Organizing Committee

You hold in your hand the proceedings for SIGBOVIK 2015. Or, more likely, you're looking at them on a screen because you're too cheap to actually buy the proceedings. Either way, we thank you for your interest, but we thank you considerably more if you've demonstrated your interest by giving us money. As is traditional, we begin the proceedings with a message that nobody will read, but which must nonetheless be included because we don't want to give any of our authors the satisfaction of having their paper start on page 1.

We are proud to announce that SIGBOVIK has experienced a surge in interest in recent years. As can be seen in Figure 1, the number of submissions grew exponentially¹ from 2012 to 2014. We don't consider the number of submissions for 2015 here, because this message comes at the beginning of the proceedings and we wouldn't want to spoil the surprise. More people have been giving presentations, and attendance has been thriving. Of course, we don't keep numeric records of these facts, because that requires organizing, and does the SIGBOVIK organizing committee look like some group of people who are interested in organizing things?

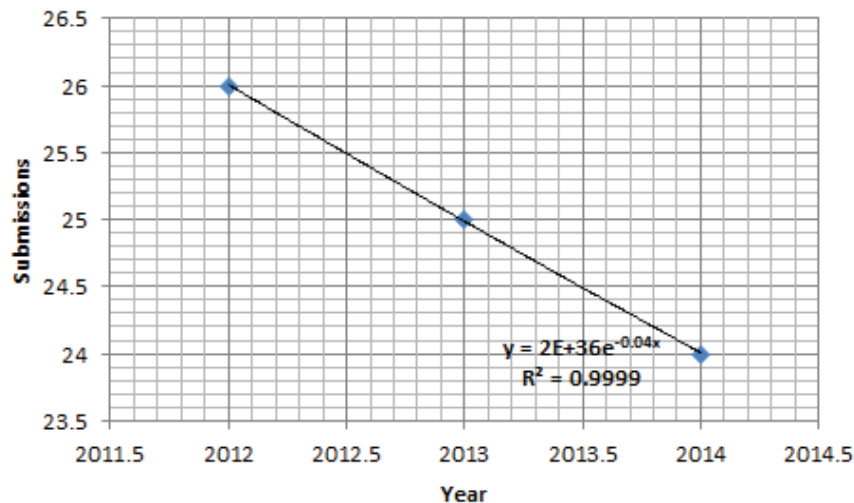


Figure 1: $y = 2 \times 10^{36} e^{-0.04x}$, where y is the number of submissions and x is the year. Yeah, this graph was made in Excel. So what?

¹Exponential decay is just exponential growth with a negative exponent, right?

Because of these vast leaps in size, SIGBOVIK has reached that point in every young conference's life where we must consider splitting the talks into tracks to save time. Because time is money and all that. However, we at SIGBOVIK feel strongly that all attendees should be able to experience the full range of exciting research that SIGBOVIK has to offer, and so it is with great pleasure that we announce several proposals for creating two parallel tracks.

Two concurrent presentations. Two presenters stand at opposite sides of the front of the room and give talks concurrently. Audience members can sit closer to whichever presenter interests them more. For example, an audience member could give 25% of his or her attention to Presenter A and 75% to Presenter B by sitting one-quarter of the way to Presenter A's side of the room. Both presenters will display their slides, because the auditorium conveniently has two screens at the front of the room. We're not sure if they can show video from two different laptops, but this seems like a technical detail.

Two concurrent presentations - in 3D! As above, two presenters stand at opposite sides of the front of the room and give talks concurrently. However, we leverage advanced 3D technology to ensure that all audience members can fully appreciate both talks. Each audience member is given a pair of colored 3D glasses. One presenter wears red. One presenter wears blue. Their slides are colored accordingly and overlaid on one screen. The audience members' brains will correctly interleave the two presentations to form a coherent narrative. If the two presentations are in different fields, this probably opens us up to all kinds of funding for interdisciplinary research. We should look into that.

Timeslicing. Did you know that before your desktop had eight cores, it had only one, but you could still run multiple programs? This was accomplished through *timeslicing*. In the same way, two presenters can give talks concurrently by alternating slides. If this granularity is too high to achieve reasonable latency, the presenters could try alternating sentences, words, syllables or phonemes. The possibilities are endless.

Question-and-answer optimization. Frequent attendees of academic talks will notice that presenters spend approximately half of their total time answering questions, either during the designated question-and-answer period at the end of the talk, or when interrupted during the talk. A further observation is that attendees of talks are generally interested in asking questions or listening to the talk, but not both. We take advantage of these observations by having Presenter A give a talk and then answer questions, as usual, and Presenter B answer questions first and then give the talk. This may seem strange, but is really no different than normal because questions typically have little or no relation to what was discussed in the talk. This format has the benefit that those who wish only to ask questions can spend the entire session doing so, and those who wish only to

hear talks can spend the entire session doing so. Additionally, both presenters are able to give their talks uninterrupted because anyone who would interrupt with questions is busy asking questions of the other presenter.

Of course, there is no reason to limit ourselves to two tracks, but the above solutions clearly generalize to n -track conferences, in the standard way. We hope to roll out one or more of these exciting developments for SIGBOVIK 2016 (not SIGBOVIK 2015, because we're lazy like that). And, with that, onto the proceedings!

Table of Contents

Track X: Locally Sourced, Lovingly Handresearched

1. inDRM: Copy Control with a Personal Touch 3
2. A Hand-Held Device for User-in-the-loop Prin-ting 13
3. Artisanal Type Theory 17
4. A Constrvctive Solvtion to the Koenigs-Pittsbvrgh Bridge Problem21

Track Y: One Paper, Two Paper, Red Paper, Blue Paper

1. Red i Removal with Artificial Retinal Networks27
2. A Proof of the Twelve-Color Theorem: Hey, Nobody Bothered Before 33
3. Visually Identifying Rank37
4. Transparency Report 45

Track L: Lunch Intermission

1. Burritos for the Hungry Mathematician 49

Track Z: Computationalizing Computation

1. A New Paradigm for Certified Code 57
2. Beyond the Halting Problem: Higher Order Infinite Loop Checkers 61
3. Bashing Haskell: Reimplementing the Parsec Library Inside the Unix Shell .67

Track W: Last Period Honors English Class

1. Programming Language Fan Fiction	73
2. Acronymy: A Bidirectional Dictionary	77
3. Another Article that Makes Bibliometric Analysis a Bit Harder	79
4. Comment: SIGBOVIK Should Ban Conclusions	83
5. The Portmantout	85

Track X

Locally Sourced, Lovingly Handresearched

1. inDRM: Copy Control with a Personal Touch

Miguel Lechon

Keywords: DRM, independent game distribution, soft cryptography

2. A Hand-Held Device for User-in-the-loop Printing

James McCann and Notreal Author

Keywords: users, loops, printing, users in loops, printing

3. Artisanal Type Theory

Carlo Angiuli

Keywords: type theory, artisans, human computation

4. A Constructive Solution to the Koenigs-Pittsburgh Bridge Problem

Greg. Hanneman and Benj. Blvm

Keywords: KBP, KPBP, pittsburgh, south side, in order to honey themselves, thanks google translate

inDRM: copy control with a personal touch

Miguel Á. Lechón*

April 1, 2015

You say “I’ll just make a copy for me and a friend”, but he’ll make one and she’ll make one and when will it end?

DON’T COPY THAT FLOPPY
MC DOUBLE DEF DP[1]

Abstract

inDRM is a digital rights management scheme. The goal of inDRM is ensuring that a small amount of human reflection accompanies the process of creating and distributing each new copy of a given piece of software. It is particularly well suited for developers that expect lots of passion but little money from their users, such as independent game developers.

Noteworthy properties of inDRM are its:

- *Broken by Design* design.
- P2P activation key generation.
- Distribution history record keeping.

Reference implementations and usage examples are available here¹.

*e-mail: miguel.lechon+inDRM@gmail.com

¹<https://github.com/debiatan/inDRM>

1 Motivation

1.1 A brief history lesson

Home taping killed music thirty years ago due to music being distributed as passive, readily cloneable data. Too little was done to remedy this, and too late[2]. Computer software, in contrast, and by its own nature, needs to be active to fulfill its intended purpose, so, through the use of DRM, video games can choose to be as passive[3], aggressive[4] or passive-aggressive[5] as they so desire. This is the main reason why we have games, but not music, today.

1.2 Danger lies ahead

The current dominant threat to the computer game industry is not so much economical as it is ecological. Shastri, Morrison and White provide a concise explanation in their recent discussion[6]:

It’s a total SNAFU! Adults buy all kinds of games but never play them. Kids have time to play but behave like monomaniacs... *Minecraft* this or *LoL* that, day in, day out!

Soon we will all realize there are no *gamers* left. And when the bubble bursts, the reeking corpses of unplayed *Steam* libraries will wash this industry away.

Indeed, according to undisclosed sources at *Valve*, the average library *backlog* (bought but unplayed games) amounts to 83% of the total number of titles each user owns[7].

1.3 Danger knocks at the door

The vast majority of game developers do not see financial return as an end, but as a way of measuring the engagement of players to their games. However, in today's prosperous economical climate, money is essentially thrown at the feet of game developers for no apparent reason, tricking them into thinking they are succeeding when, in all likelihood, no one is playing their creations.

A recent example is the case of the *Handmade Hero* project[8], where one programmer with no previous history of ever completing a commercial video game announced his plans of working five hours *per week* on a yet-to-be-started title. He successfully collected thousands of preorders during the course of the following weeks.

Will this summer last much longer? Won't some brilliant academic mind devise a clever way of avoiding this impending disaster? How can cautious game developers gauge the *real* commitment of their user base when all signs indicate they are doing a splendid job?

The answers to these questions are *no*, *yes* and *read on*.

1.4 Fear no more

The solution, of course, lies in the repurposing of the industry's proven old savior, DRM.

As long as money keeps growing on the trees of developer's backyards, the amount of game copies sold will be an inaccurate measure of consumer acceptance. Instead, I propose making players spend an ultimately much more valuable resource than money: their time. In order to achieve this, and as a condition to unlock the game, the user will be forced to engage in a short social exchange with either the creator of the game or some other fellow player.

By slightly inconveniencing potential clients, creators can expect to gain a clearer understanding of the appeal of their games and end up with a much more committed, albeit smaller, user base.

2 Cryptological interlude

inDRM's strength rests on:

- the solid foundation of the MD5/4 cryptographic hash function
- the RSA public key cryptosystem signing procedure [10] with a key length of 32 bits
- the identification of computers via the MAC addresses of their primary network interfaces.

This section consists of a rather in-depth review of the first two algorithms.

2.1 MD5/4

MD5[9] takes an arbitrary string of characters, passes it through a deterministic blender and produces an unnecessarily long 128-bit value.

MD5/4 takes that value, chops it into four 32-bit pieces and XORs them together to produce a more digestible 32-bit digest.

2.2 RSA signatures

The RSA signature procedure is a popular party trick from the late 1970s, where one person comes up with three numbers n , d and e that satisfy the following property:

$$(x^d)^e \equiv x \pmod{n}, \forall x \in \mathbb{Z}^2$$

That person then makes n and e public, spends a few minutes teaching modular exponentiation to the crowd and claims to possess the ability to:

- Turn any message into a number x (using a scheme similar to MD5/4)
- Produce a number y that depends on x and that acts as that person's signature of the message, fact which can be verified by checking that: $y^e \equiv x \pmod{n}$

Of course, behind the scenes, the entertainer obtains y by raising x to the d th power, dividing the result by n and taking y to be the remainder of that division.

The process of finding the three initial numbers is demonstrated in [11] and can be

² Which translates into Pythonist parlance as:
`(pow(pow(x,d,n),e,n) == x) == True`
for any integer value of x

easily accomplished today with the help from the time-tested software package `openssl`[12] by issuing the following command in any POSIX-compliant operating system:

```
openssl genrsa 32|openssl rsa -text3
```

Looking at its output, the `modulus` field is equivalent to our n , `publicExponent` denotes e and `privateExponent` indicates d .

3 Guided tour

Any person in possession of an unlocked instance of a game protected by `inDRM` can unlock copies for other potential players, as long as those copies descend, directly or indirectly, from that same unlocked instance.

There are several distinct phases in the distribution of a game protected by `inDRM`:

- Author **A** of a game generates a *root activation key file* and distributes it along with the game
- Potential player **P** obtains the game, tries to run it and is invited by the software to generate a *request file* and send it back to author **A** as a precondition for playing
- Author **A** receives the *request file*, throws a small party and generates a *response file* for player **P** that becomes a valid *activation key file*
- Player **P** plays the game, finds it worthwhile and hands a copy to friend **F**

³It is fashionable to use 2048 instead of 32, probably in reference to the homonymous video game[13]

- Friend **F** tries to run the game and is invited by the software to generate a *request file* and send it back to either **A** or **P** as a precondition for playing. **F** chooses to contact **P**
- **P** generates a *response file* for friend **F**, by virtue of possessing a valid *activation key file* belonging to the same *key chain* that reached **F**
- ...

The rest of this section reviews the technical details behind the generation of key files.

3.1 Root activation key file

The creator of the game generates a root activation key file and distributes it with every copy of the game. In and of itself, that key file only allows the game to be played on the developer's computer. However, it provides the basis for the generation of key requests from potential players.

Here is an example of an inDRM root activation key file:

```

===== inDRM key file =====
game: Adventures in inDRMland
=====
nick: debiatan
location: Barcelona
date: 2015/04/01
notes: Enjoy!
mac_salt: 1e6c40ea
mac_hash: 76f0da12
hash: c738b172
signature: d38cc9a

```

Inside key files, lines starting with an equal sign are ignored. The rest are composed of a tag, followed by a semicolon and a value field. The tags and their associated functions are these:

- game:** Title of the game
- nick:** Author's name or nickname
- location:** Author's place of residence
- date:** Date of creation of the key file (in *yyyy/mm/dd* format)
- notes:** Notes from the author
- mac_salt:** Random 32-bit hex number
- mac_hash:** MD5/4 hash of the concatenation of **mac_salt** and the hexadecimal representation of the MAC address of the primary network interface of the computer generating the key file
- hash:** MD5/4 hash of the concatenation of: the preceding **signature** in the key chain (assumed to be "0" in case of the master key file), **game**, **nick**, **location**, **date**, **notes**, **mac_salt** and **mac_hash**
- signature:** RSA signature of **hash** (computed as $(\text{hash}^d \bmod n)$, where d and n have been generated along with e as discussed in section 2.2).

For the particular example used in this section, the values of the cryptographic parameters are:

- n (modulus): 3333098473
- e (public exponent): 65537
- d (private exponent): 939472245

These values are to be embedded in the `inDRM` routines present in the game. The pair (e, n) will be used as an RSA public key in order to check signatures present inside key files. The pair (d, n) will allow a registered copy of the game to sign key file requests from new potential players through an in-game menu option labeled to that effect.

3.2 Activation key file validation

Every time a piece of software guarded by `inDRM` is run it reads the activation key file and checks that:

- the value of the `hash` field is correct (by recomputing it using the tag values that precede it)
- the value of the `signature` field is correct (by ensuring that $\text{signature}^e \bmod n = \text{hash}$)

If these two conditions are met, the primary MAC address of the machine is checked to see if it satisfies:

$$\frac{\text{MD5}}{4}(\text{mac_salt}_i + \text{MAC}) = \text{mac_hash}_i$$

(where the ‘+’ sign indicates concatenation of strings) for any `mac_salti/mac_hashi` pair present in the file. If it does, the game is allowed to run. Otherwise, the user is invited to generate a request file.

3.3 Request file generation

An invitation to generate a request file will essentially convey a message similar to this one:

```
*** Last MAC address on key file does
    not belong to this computer ***
You won't be able to play this game
unless you convince another player to
generate a key for you.
```

```
Let's build a request file...
```

```
Please provide the following data
(or press 'enter' to skip):
Name (or nickname): _
Location (place of residence): _
Notes (message to future players): _
```

After providing (or failing to provide) the three pieces of information, `inDRM` collects the date and MAC address of the system, generates the request file and tells the user that:

```
A request file has been generated here:
*** /home/ ... /request.txt ***
```

In order to finish the registration process, send that file back to whoever shared the game with you. That person will be able to unlock your copy.

Think twice before sharing this game with other people. If they ever try playing it, they might come back asking you to register their copies.

The request file consists of a copy of the original key file distributed with the game with an extra section appended at the end. Imagining that the original key was the one presented back in section 3.1 and the user provided “miguel”, “barcelona” and “this

sucks” as values for **name**, **location** and **notes**, respectively, the resulting extra section could look like:

```
nick: miguel
location: barcelona
date: 2015/04/01
notes: this sucks
mac_salt: 95be1f47
mac_hash: 6051a20a
hash: 1f495ce2
signature: NO SIGNATURE YET
```

In order to run the game, the potential player will then send the request file up the distribution chain for it to be signed.

3.4 Response file generation

The example activation chain we have described consists of only the original author of the software and its first user. The request file will then be necessarily sent to the author, who will execute a registered copy of the game and select the option that allows to sign requests. That routine will check that:

- the values of all **hash** fields are correct (by recomputing them using the values of fields preceding it)
- the values of all **signature** fields, except for the last one, are correct (by ensuring that $(\mathbf{signature}^e \bmod n = \mathbf{hash})$)
- the primary MAC address of the computer satisfies:
 $\frac{\text{MD5}}{4}(\mathbf{mac_salt}_i + \text{MAC}) = \mathbf{mac_hash}_i$
for some $\mathbf{mac_salt}_i/\mathbf{mac_hash}_i$ pair in the request file.

If all three conditions are met, the last **signature** field of the request file will be completed with the help of the private exponent **d** and the modulus **n**:

$$\mathbf{signature} = \mathbf{hash}^d \bmod n$$

In our example, this would mean modifying the last **signature** of the request file to read:

```
signature: 9bc727d0
```

The resulting file will then be sent back to the potential player to be used in place of the original activation key file. The *potential player* will then stop being just *potential* and start a new life as a *real player*.

4 Properties

Improving security typically means degrading usability. Providentially, both effects are intended consequences of the use of **inDRM**.

There are several other desired properties that have gone into the design of the **inDRM** copy control scheme. This section discusses them briefly.

4.1 *Broken by Design* design

If we were to depict the concept of security as a one-dimensional horizontal segment delimited on the left by the word *insecure* and on the right by the word *secure*, **inDRM** would lie definitely to the left, in a place probably labeled as *cryptographically annoying*. Sadly, space limitations preclude even a cursory exploration of all the security flaws exhibited by **inDRM**.

4.2 P2P key generation

Players of games protected by **inDRM** can rest assured that they will be able to enjoy them for as long as they are interested in doing so. The absence of network authentication and the distributed nature of the key generation process guarantee that.

Moreover, as a side benefit of the distributed nature of the key generation scheme, players might derive some sense of belonging to a community by simple inspection of the contents of their personal key files. Intensely imaginative players, of the kind that take pleasure in trading *Steam* inventory items, may even experience **inDRM**'s compulsory chaining of signatures as a tradition that makes them part of a lineage.

4.3 Lack of attractive as a cracking target

All DRM schemes are eventually subverted, so one should not ask *if* but *when* will **inDRM** be broken. I suspect that it will not happen any time soon for mainly two reasons:

- Breaking **inDRM** poses no challenge, so no one can take pride on that endeavor
- The only benefit to be derived from breaking **inDRM** is the avoidance of a short social interaction

4.4 Distribution history record keeping

Key files have potential uses other than game distribution control. For instance, the com-

bined **notes** sections of a key can be used as a simple guestbook or, perish the thought, a space for advertisements.

Another more elaborate use requires us to picture the set of activation key files associated to a particular game as a tree, its *leaf* key files being the ones that carry the most information. Getting hold of a sizeable percentage of those files by some means⁴, would facilitate the identification of the principal hubs in the game's distribution network.

4.5 Simplicity

MD5 is straightforward to program[14]. The improvements that MD5/4 introduces come at the cost of three extra XOR instructions.

Implementing the standard RSA algorithm would require working with arbitrarily long integers, but the 32-bit version of it does not impose that hardship on game developers. Two additional benefits of working only with 32-bit values are that key files become smaller in size and that they fit inside the two-column layout of this article.

4.6 Involvement of players

In order to acquire a valid activation key file, the potential player must secure the collaboration of another player. After implicating that other person, the new player may feel pressured to give the game a proper try and avoid dropping it after just a few minutes of play.

⁴such as asking nicely or offering players extra game content in exchange for their keys

4.7 Future-proof architecture

As long as one registered copy of the game remains, it will continue to be redistributable and playable. Even if no registered copy remains and even if the game binary proves difficult to crack, bruteforcing the **signature** of a request file requires just the testing of the set of 2^{32} possible signatures.

5 Conclusion and final remarks

Game developers do not need protection from software pirates; they need instead to be guarded from the uncaring capricious money-throwing player hordes that endanger their profession. By choosing to distribute games guarded by inDRM, they are not only deciding to acquire a much more mature following, but are also telling the video game community that *they care*.

For an up-to-date list of games that use inDRM, check:

<http://blog.debiatan.net/inDRM.html>

References

- [1] MC Double Def Disk Protector, *Don't copy that floppy*⁵, 1992
 - [2] *Sony BMG copy protection rootkit scandal*⁶, Wikipedia
 - [3] *Good Old Games*⁷
 - [4] *Always-on DRM*⁸, Wikipedia
 - [5] *DRM – Software tampering*⁹, Wikipedia
 - [6] S. Shastri, R. T. Morrison, and X. White, *More games, less time*. Journal of Bad Omens, vol. 35, pp. 75-80, Dec. 2012.
 - [7] “Undisclosed” means “undisclosed”.
 - [8] *Handmade Hero*¹⁰
 - [9] R. Rivest, *The MD5 message-digest algorithm*, RFC 1321, April 1992.
 - [10] R. Rivest, A. Shamir and L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM 21 (2): 120–126. February 1978.
 - [11] *RSA encryption and decryption, a worked example*¹¹, Wikipedia
 - [12] *OpenSSL*¹², Wikipedia
 - [13] *2048*¹³, Wikipedia
 - [14] *MD5 pseudocode*¹⁴, Wikipedia
-
- ⁵<http://www.gog.com/>
⁶https://en.wikipedia.org/wiki/Always-on_DRM
⁷https://en.wikipedia.org/wiki/Digital_rights_management#Software_tampering
⁸<https://handmadehero.org/>
⁹[https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)#A_worked_example](https://en.wikipedia.org/wiki/RSA_(cryptosystem)#A_worked_example)
¹⁰<https://www.openssl.org/>
¹¹[https://en.wikipedia.org/wiki/2048_\(video_game\)](https://en.wikipedia.org/wiki/2048_(video_game))
¹²<https://www.openssl.org/>
¹³[https://en.wikipedia.org/wiki/2048_\(video_game\)](https://en.wikipedia.org/wiki/2048_(video_game))
¹⁴<https://en.wikipedia.org/wiki/MD5#Pseudocode>



CONFIDENTIAL COMMITTEE MATERIALS

SIGBOVIK 2015 Paper Review

Paper 1: *in*DRM: copy control with a personal touch

Chris Martens

Rating: 2 (destroy all games)

Confidence: 4/4

The copy protection scheme described in this paper ostensibly poses a solution to a threat to the video games industry. As such, I cannot sanction it as a good idea. Also, the paper is eight pages long and had a lot of boring stuff about crypto in it.

On the plus side, the “Broken by Design” design seems like a useful feature.

A Hand-Held Device for User-in-the-loop Printing

James McCann*
TCHOW

Notreal Author
WeAreWe University



Figure 1: Our handheld hardcopy output device (left) shares many of the capabilities of a commercially produced printer (right), but in a much more convenient form-factor.

Abstract

Thanks to the proliferation of the ARPANET, computers are used every day by hundreds of people around the world to access a wealth of information. However, until recently, this information could only be applied in domains that could be brought within reach of a computer terminal, or by operators with extensive memory training.

With the advent of the *printer*, this has begun to change. Computer operators now are able to send digital information to a machine which converts it into *hardcopy*, a physical representation constructed from marks on paper. This hardcopy information can then be transported to application locations distant from any computer access terminal and applied.

While this process is certainly more user-friendly than the alternatives of relying on memory or movement of information application domains, it remains cumbersome, expensive, and inaccessible. We propose a hand-held device which can produce informational markings on paper, much like a printer. Hardcopy produced with our device has similar information content to that produced with a conventional printer, but at a lower price.

Further, we describe *user-in-the-loop printing*, a process enabled by our device, which allows computer operators to filter and transform information as it flows from the computer to the hardcopy. These transformations can lead to ink savings through domain-tailoring. We also demonstrate how our device can allow computer operators to increase the information output of existing hardcopy – either from a conventional

printer or our own device – through a process we term *information overlay*.

CR Categories: B.5.2.s [Rock]: Rock—Lobster

1 Introduction

ARPANET [Davies et al. 1967] is an inter-network of computers, which are able to exchange information between themselves using digital network signalling. Estimates vary, but most experts [Callahan and Hodgman 1971] agree that between tens and hundreds of people use the ARPANET every week, transmitting – in aggregate – more than 400 *kilobytes** of data during the average month.

However, until recently [2014], this information was confined to locations with direct ARPANET access, such as “ARPANET Cafes,” or to those locations to which computer operators with exceptional memory skills could travel, such as “Cafes.” However, with the advent of *printers*, this restriction has been relaxed.

In this paper, we extend the notion of *printing* with a hand-held hardcopy output device. Our device provides many of the same output capabilities of a full-scale *printer*, without requiring a movable type department or access to steam power (a significant cost savings). Further, our handheld device enables *user-in-the-loop printing*, through which the computer operator can actually transform the information which they record.

*A *kilobyte* is unit of information equal to the storage capacity of 64 *kilograms* of quarters (practitioners often summarize this by saying there are “sixteen bits in one byte”.)

*e-mail: ix@tchow.com

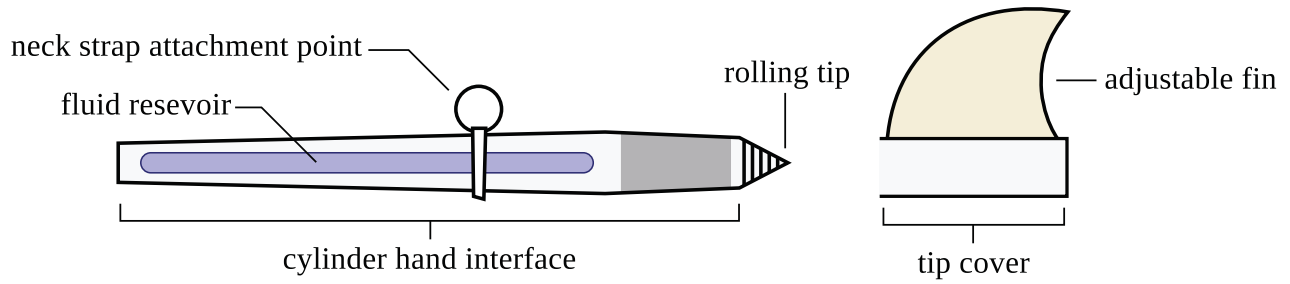


Figure 2: A labeled diagram of our hardcopy output device. For more information see the text.

2 Our Tool

A prototype of our hardcopy output device is shown in Figure 1, with a schematic diagram shown in Figure 2. We briefly explain the construction here. Curious readers are invited to submit an NDA request to the second author to be granted access to further details.

The operative end of the device is the *rolling tip*, which distributes ink from the *fluid reservoir*. In non-printing situations, the rolling tip is protected by the *tip cover*, which is also the anchor point for the *adjustable fin* – which we believe contributes to the overall aerodynamics of the device.

The fluid reservoir is surrounded by the *cylinder hand interface*, which – as the name suggests – is the control surface through which the device is actuated during printing operations.

Finally, the *neck strap* is an ergonomic addition aimed at mitigating hand fatigue during long printing sessions.

3 Basic Operation

During a basic printing session, the operator first secures the neck strap to the device and adjusts it for optimal comfort. The operator then transfers the tip cover from the rolling tip end of the device to the tip cover storage location on the non-tip end of the cylinder hand interface. Next, the operator secures a piece of printing substrate to their video ARPANET computer terminal device, and – controlling the rolling tip’s position with the cylinder hand interface – directs our device to move along the printing substrate in the patterns visible through it.

This has the effect of creating ink deposits on the substrate, registered to the ARPANET information the operator wishes to capture.

Though this process may sound complicated, we find that most operators become proficient within the first 1-2 weeks of incarceration in our specialized training camp, at which point they are allowed to return home.



Figure 3: User-in-the-loop printing allows users to excerpt just the information they require from a document (*sinister*), unlike conventional printing, which reproduces the entire text (*dexter*).

4 User-In-The-Loop Printing

Our hardcopy output has many advantages over a commercial printer. One large advantage is that – because the user is controlling the device – they are able to transform the information that is recorded. We call this process *user-in-the-loop printing*.

In this section, we summarize the *printing* actions taken by a group of computer operators using our device. We recruited these operators through an ARPANET posting, and in no way threatened or cajoled them into participating. We list only the most common operations, since we think that a more detailed list is probably worth saving for a second publication.

4.1 Condensation

One of the most common transformations we observed was the removal of extraneous textual information. This information removal allowed users to speed up their hardcopy pro-

duction process by omitting all but the most pertinent details.

Often this would involve the elimination of entire paragraphs and sections, save for a few key equations or phrases. This is a huge material and time savings that cannot be accomplished by commercial printers.

4.2 Abstraction

Abstraction is the graphical analogue of condensation. Often, when tasked with producing hardcopy of a particular image, operators would remove much of the detail from the image; reproducing an approximation with only a few strokes.

As with condensation, we found that the level of detail reproduced was often task-dependent. For instance, when producing hardcopy of a map, users were likely to carefully reproduce streets near their intended journey or destination, and simplify or elide streets elsewhere.

4.3 Spelling Alteration

During the course of hardcopy production, we often saw users change the spelling of words. We believe that this is a positive indication that our handheld printing device can contribute to the evolution of our living language.

4.4 Incremental Refinement

Finally, several of our participants used an unexpected, but exceedingly efficient, printing strategy, wherein a document produced on a traditional printer was refined using our hardcopy output device.

Though this hybrid output scenario was unexpected, we found that its unexpected unexpectedness was not something we expected, but we would not expect that follow-up work in this unexpected direction would be unfruitful. So such follow-on work shouldn't be unexpected, unlike this unexpected behavior, which wasn't something that was not unexpected.

5 Limitations

Our handheld printing device does have a few limitations, as compared to a traditional printer. But none worth mentioning.

6 Conclusion

In this paper, we described a revolutionary device that enables computer operators to produce hardcopy without the need for an expensive printer. In testing, we observed that operators used our device not just to produce exact copies of documents, but actually significantly transformed those copies ($p < 2.0$). We termed these transformations *user-in-the-loop printing*.

We have only tested our hand-held printing device on flat surfaces; however, we are excited about the possibility of recording information on non-flat surfaces – a process we term “3D printing”. We believe *user-in-the-loop 3D printing* has the potential to revolutionize the decorative arts.

References

2014. A recent year.

CALLAHAN, E., AND HODGMAN, J. F., 1971. John Hodgman.

DAVIES, D. W., BARTLETT, K. A., SCANTLEBURY, R. A., AND WILKINSON, P. T. 1967. A digital communication network for computers giving rapid response at remote terminals. In *Proceedings of the First ACM Symposium on Operating System Principles*, ACM, New York, NY, USA, SOSPP '67, 2.1–2.17.

A Even Prime Numbers

$$p_{\text{even}} = \{2\} \tag{1}$$

B Integral Roots of Unity

$$\{n \mid n \in \mathbb{Z}, n^2 = 1\} = \{-1, 1\} \tag{2}$$

C Composite Prime Numbers

$$\emptyset \tag{3}$$

Artisanal type theory

Carlo Angiuli

April 1, 2015

1 A brief history of some things

Food was invented by Mesopotamians some 5,000 years ago, and has been eaten ever since. Logic was invented in the Mediterranean by ancient Greeks, including Aristotle and a mortal man [1] named Socrates. It lives on as an important course in pre-law curricula across the United States.

Modern times require more modern logics. Computer programming is closely tied to *intuitionistic* logic, in which proofs of a proposition correspond directly to algorithms. Intuitionistic logic, often in the form of type theory, is taught to several American computer scientists annually.

Until the past several centuries, food and logic were primarily manufactured by artisans, who trained apprentices in the arts of, respectively, proofing and proving. The Industrial Revolution gave rise to machines able to produce food and textiles much faster than artisans ever could. The digital revolution, likewise, has turned ‘computer’ from a human job into a cheap, ubiquitous machine capable of multiple calculations per second.

Despite the overwhelming success of mass-produced food, some consumers want to revisit food’s roots as a product sustainably and ethically crafted by local artisans using traditional techniques. The result, known as *artisanal food*, has taken off in popularity in the past few years [2, 4].

2 Algorithms with the human touch

So too should computer scientists return to the roots of computation—slow, error-prone calculations performed by humans. After all, despite the close relationship between computer programs and type theory [3], or indeed, between computer programs and algorithms, there is no need to involve computers in deeply human tasks like sorting lists, routing packets, or decoding MPEG-4 video streams.

We advocate a more personal approach to computation, called *artisanal type theory*. Artisanal type theory has the same rules as ordinary type theory, except that all terms and typing derivations must be handwritten. Closed, well-typed terms evaluate to values of the same type, accompanied by a certificate of authenticity that a human performed that evaluation.

Since each term was lovingly handcrafted and normalized, these artisanally-performed beta-reductions provide a more *meaningful* explanation of type theory than traditional computer-based interpreters. Using artisanal type theory demonstrates a firm commitment to *locally*-grown, sound, and complete reasoning systems.

3 Examples

Shallow learning. We can implement artificial intelligence using human intelligence. Given the training data that `false` is desired but `true` is not, one can manually build a classifier for booleans. Such a classifier can then be run on a boolean to determine whether or not it is in the desired set. In the example below, we run the classifier on `false`.

$$\begin{array}{c}
 \frac{x:\text{bool} \vdash x:\text{bool} \quad x:\text{bool} \vdash \text{false}:\text{bool} \quad x:\text{bool} \vdash \text{true}:\text{bool}}{x:\text{bool} \vdash \text{if } x \text{ then false else true} : \text{bool}} \\
 \frac{\bullet \vdash \lambda x:\text{bool}. \text{if } x \text{ then false else true} : \text{bool} \rightarrow \text{bool} \quad \bullet \vdash \text{false} : \text{bool}}{\bullet \vdash (\lambda x:\text{bool}. \text{if } x \text{ then false else true}) \text{ false} : \text{bool}} \\
 \equiv [\text{false}/x](\text{if } x \text{ then false else true}) \\
 \equiv \text{if false then false else true} \\
 \equiv \text{true}
 \end{array}$$

LOVINGLY REDUCED
 BY CARLO ANGIULI
 3/16/2015

Fairly quick sorting. Artisanal computation has some advantages over computer programming—namely, humans can perform some computations without needing a precise algorithm. In this example, we sort a list of integers without the need to specify a particular sorting algorithm.

$$\begin{array}{c}
 \frac{\text{I can do that} \quad \text{trivial?}}{\bullet \vdash \text{sort} : \text{int list} \rightarrow \text{int list} \quad \bullet \vdash [3, 7, 9, -1, 4] : \text{int list}} \\
 \bullet \vdash \text{sort } [3, 7, 9, -1, 4] : \text{int list} \\
 \equiv [-1, 3, 4, 7, 9]
 \end{array}$$

LOCALLY EVALUATED
 IN PITTSBURGH
 3/16/2015

Small-batch jobs. We can also write scripts which artisanally perform repetitive computing tasks such as renaming a large number of files, accessing sequential URLs, etc. Doing so requires adding primitives for I/O, file system access, and so forth. As discussed above, it is unnecessary to actually implement these primitives, because the human runtime already supports these tasks via a computer’s traditional user interface. Furthermore, unlike in traditional scripting languages, it is easy to extend this with physical-world primitives such as `eatLunch`, `writeHandwrittenThankYouNote`, and `sleep`.

4 Conclusion

We hope artisanal type theory will appeal to computer scientists and mathematicians who appreciate algorithms but believe computers themselves are quite inconvenient at times. Unlike ordinary type theory, it directly expresses computations as if people, but not computers, matter. Therefore, we are hopeful that artisanal type theory will be a useful foundation for the field of *human science*.

References

- [1] ARISTOTLE. *The Organon*. 40 B.C. Ed. Andronicus of Rhodes.
- [2] COPE, S. *Small Batch: Pickles, Cheese, Chocolate, Spirits, and the Return of Artisanal Foods*. Rowman & Littlefield Publishers, Inc., 2014.
- [3] MARTIN-LÖF, P. Constructive mathematics and computer programming. In *Proc. Of a Discussion Meeting of the Royal Society of London on Mathematical Logic and Programming Languages* (Upper Saddle River, NJ, USA, 1985), Prentice-Hall, Inc., pp. 167–184.
- [4] SCHWANER-ALBRIGHT, O. Brooklyn’s new culinary movement. <http://www.nytimes.com/2009/02/25/dining/25brooklyn.html>, Feb. 2009.

A CONSTRUCTIVE SOLUTION TO THE KÖNIGS-PITTSBURGH BRIDGE PROBLEM

Greg. HANNEMAN
Language Enthusiast and Errant Scribbler
Carnegie Mellon University

Benj. BLVM
Light Cone Sedentarian
Carnegie Mellon University

ABSTRACT.

We walked across a bunch of bridges one day.

§1..... BACKGROVND.

Figure 1 and Definition 1 state the well-studied Königsberg Bridge Problem [Eul41].

Definition 1. The Königsberg Bridge Problem (KBP).

Regiomonti in Borussia esse insulam A der Kneiphof dictam, fluviumque eam cingentem in duos dividi ramos, quemadmodum ex figura videre licet: ramos uero huius fluvii septem instructos esse pontibus, a, b, c, d, e, f, et g. Circa hos pontes iam ista proponebatur quaestio, num quis cursum ita instituere queat, ut per singulos pontes semel et non plus quam semel transeat.

According to the best modern scholarship, it is thought that this passage reads as follows.

„At Königsberg in Prussia there is an island A known as Kneiphof, and flowing around it is divided into two branches, as shown in the figure; and the branches of the river, seven bridges, however, of this, \$a\$, \$B\$, \$C\$, \$d\$, \$e\$, \$F\$, and \$G\$. Already presented to him, the question concerning these bridges is, the course to be instructed so to whether a person may be able to, and not more than once in order to pass one by one the bridges honey themselves.”

The criteria expressed in Definition 1 have been generalized to arbitrary geographical

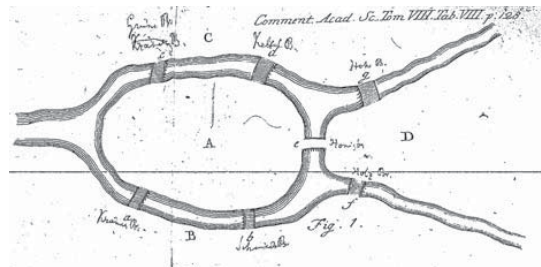


Figure 1: Kaliningrad, Russia (colloq.); better Known in the Formal Literature as Königsberg, Prussia

topologies. Of extreme local interest is the instance derived from the city of Pittsburgh, Pennsylvania, V.S.A., as Definition 2.

Definition 2. The Königs-Pittsburgh Bridge Problem (KBPB).

Cross every pedestrian-navigable trans-Three-River bridge with at least one endpoint in the city of Pittsburgh, exactly once, to be accomplished during one continuous circuit.

This article presents a constructive proof for the existence of a solution to KBPB. Furthermore, while most prior work [Wal15] has largely focused on theoretical analysis of KBP variants based on graph theory, here we additionally offer a more practical approach based on ambulation theory.

§2..... FORMALIZATION.

The formal statement of KBP inquires after the existence of an Eulerian path on a graph representing Königsberg, with one node to repre-



Figure 2: informal Diagram of the Topology of Pittsburgh, Pennsylvania as defined by its Bridges and Rivers.

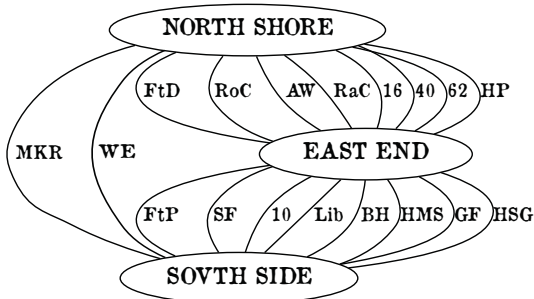


Figure 3: Formalization of Figure 2.

sent each of the landmasses connected by various edges (bridges). In KPBP we simply change the graph to represent the topology of our home city (Figure 2).¹

Unlike KBP, in KPBP we require an Eulerian circuit, rather than an Eulerian path. Recent developments in ambulation theory have shifted the common wisdom towards a preference for the use of circuits over paths in ambulatory proofs, as the circuit presents a more convenient practical application. (Put in more intuitive terms, the Eulerian circuit allows the bridge-walkers to sleep in their own beds the following night, rather than being stranded across a river.) Fortunately, the topology we have selected for KPBP allows for such a solution.

¹The astute observer will note that no edge among our formalism represents the 32nd street bridge. During the progress of our research, this bridge did not exist due to construction, and hence is omitted. If added to the graph (call it KPBP'), it is known that a non-cyclic solution exists, but a constructive proof is left to future work.

§3..... PROOF.

Theorem 1. The Königs-Pittsburgh Bridge Problem is solvable.

We present here a constructive solution to the problem, first given by Hanneman, Vangpat, and Blum (2012, Oct. 27, 5:15 a.m.–10:05 p.m.). The full solution has length 35 miles and consumes approximately 17 hours of wall-clock time. We depict R in two distinct styles, in a somewhat abbreviated form, in Table 1 and Figure 4.

Intermediate details may easily be filled in by following Procedure 1.

Procedure 1. Let R be the given solution route for KPBP, p be a position along it measured in miles from the start of R , and \hat{r}_x be a unit vector in the direction of R at position x .

```

while  $p < |R|$  do
   $\vec{d} \leftarrow \text{StepSize} \cdot \hat{r}_p$ 
  Walk( $p, \vec{d}$ )
   $p \leftarrow p + \|\vec{d}\|$ 
end while

```

Remark 1. Some experimental determination will of course be necessary to arrive at the correct value of the StepSize variable. We suggest a default value of 75 cm, to be adjusted upwards or downwards as personal height and local terrain require.

Lemma 1. Walkability.

A point p' along R can be reached from point p along R , following Procedure 1, assuming $p < p'$.

Proof. Try it. □

Lemma 1 thus establishes a “modus ponens” result for traversal of the route R from its beginning ($p = 0$) to its end ($p = |R|$). This is, however,

Node	Edge	Node	Node	Edge	Node
EE	HP	NS	NS	FtD	EE
NS	62	EE	EE	FtP	SS
EE	40	NS	SS	SF	EE
NS	16	EE	EE	10	SS
EE	RaC	NS	SS	Lib	EE
NS	AW	EE	EE	BH	SS
EE	RoC	NS	SS	HMS	EE
NS	MKR	SS	EE	GF	SS
SS	WE	NS	SS	HSG	EE

Table 1: $|R| = 35$ mi.



Figure 4: a cartographic Rendering of the Solution Route R.

a purely theoretical result. We now show the practicability of the solution in an operational scenario via a number of satisfiable conditions.

Condition 1. Initialization.

Let A be a set of attempting participants. Then a must be present at $p = 0 \forall a \in A$.

Condition 2. Progression.

Let B be a set of qualifying bridges such that the following are true $\forall b \in B$: (1) b satisfies the bridge criteria from Definition 2; (2) b has defined $0 \leq p_b \leq 35$ on R . Let b_i be the i th ordered element of B , and let $|B| = n$. Then $C = \{c_1, \dots, c_n\}$, and $c_i = (S_i, b_i)$ iff some $S_i \subseteq A$ crosses b_i .

Condition 3. Termination.

Let the set of ending participants $E \subseteq A = \{a \in A \mid a \in S_i \forall i\}$. Then the solution is successful iff $E \neq \emptyset$ and e is present at $p = 35 \forall e \in E$.

Lemma 2. Satisfiability of Conditions.

Conditions 1–3 are satisfiable.

Proof. Set $A = \{\text{Alan, Ben, Dan, Greg, Keith}\}$. Note that $\Pr(x \text{ awake at 5 a.m.}) = \epsilon \forall x \in A$. As long as $\epsilon > 0$, $\epsilon^5 > 0$. [VII14]

Set $B = \{\text{Highland Park, 61st Street, 40th Street, 16th Street, 9th Street, 7th Street, 6th Street, McKees Rocks, West End, Fort Duquesne, Fort Pitt, Smithfield Street, Liberty, 10th Street, Birmingham, Hot Metal, Glenwood, Homestead Grays}\}$. The bridge criteria of Definition 2 are satisfied $\forall b \in B$. Therefore the set C

of crossings may be freely constructed following R from Figure 4.

Finally, set $E = \{\text{Alan, Ben, Greg}\}$. The diagrams in Figure 5 offer a visualization of these values for A , B , and E .

Since $|E| = 3 > 0$, this proves Lemma 2 and thus the successful instantiation of Conditions 1–3. Together with Lemma 1, this proves that R is a navigable Eulerian circuit of the city of Pittsburgh. Therefore a solution to the Königs-Pittsburgh Bridge Problem exists. \square

§4..... CONCLVSION.

Vetere Königsberg consequat pons (KBP) designat insulae urbem, flumina, pontes, qui in tali figurazione transiens per singulos semel iter unum non potest. In hoc articulo quaestionis exposuimus accommodatio hodierna die ad oppidum Pittsburgh (KBPB). Significat adaptationem ad profectum nostrum insignis civitas-in-es super duobus.

Vno modo, sicut KBPB est solubilis, offert mathematici nostram collationem incitamentum ad ire foris, et accipiam ab eo aliquid exercitium, quod hodie magis quam somnium, quod fieri non potest. Deinde insuper offerebat speculativa ratio nos instruit etiam problema aedificant, qui probi ad ultimum pedes decem milia multa mala!

REFERENCES.

- [Eul41] Leonh. Eulero. Solvtio problematis ad geometriam sitvs. In Commentarii academiae scientiarum Petropolitanae pp. 128-140, 1741.
- [Van12] Alan Vangpat. Alan's photos. <http://journal2.alanv.org/?p=2675>, 2012.
- [VII14] Dr. Tom Murphy VII. What, if anything, is epsilon? In Proceedings of the Eightieth Centennial Intracalary Robot Dance Party in Mourning of Harry W. Bovik's 3⁶th Birthday, 2014.
- [Wal15] James Wales. Seven bridges of königsberg. In Wikipedia, the free encyclopedia, 2015.



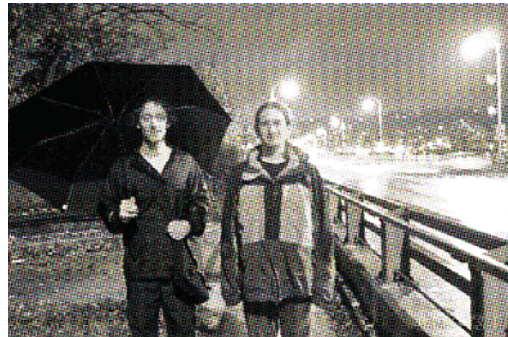
(a) Condition 1, Initialization.



(b) an example Member of the set B.



(c) $\exists b \in R$ s.t. $b \notin B$.



(d) Condition 3, successful Termination.

Figure 5: Satisfiability of Conditions. [Van12]

Track Y

One Paper, Two Paper, Red Paper, Blue Paper: and Other Fun Games for Children

1. Red i Removal with Artificial Retinal Networks

Dr. Tom Murphy VII Ph.D.*

Keywords: computational photography, image processing, generalized photoshop, artificial retinal networks, types

2. A Proof of the Twelve-Color Theorem: Hey, Nobody Bothered Before

Y. T. Boveck and Victoria V. Varg-Mack

Keywords: twelve colors, planar giraffes, time travel, drivel, fonts are ugly in the future

3. Visually Identifying Rank

David Fouhey, Daniel Maturana, and Rufus von Woofles

Keywords: perceptual organization, vitamin rank deficiencies, egalitarianism in the PSD cone, PAC bounds for SVDs, class-conscious norms

4. Transparency Report

Three Letter Agency

Keywords: transparent, invisible, see-through

Red i removal with artificial retinal networks

Dr. Tom Murphy VII Ph.D.*

1 April 2015

Abstract

We present a GPU-accelerated means for red i removal in photographs.

Keywords: computational photography, image processing, generalized photoshop, artificial retinal networks, types

1 Introduction

When light—such as the bright flashbulb of a camera—strikes the human eye, it illuminates the retina. Some of that light bounces back out of the eye, but most of it stimulates neurons in the retina to produce electrical signals. These signals stimulate other neurons to which they are connected, and so on, until the brain (which is technically part of the eye) perceives an image, as a two-dimensional array of neurons with different activation levels. Humans often use these images to sense the world, for example, in reading research papers.

This research paper concerns a particular feature of this process, which is that humans are able to view an image and ignore certain details of it. For example, Figure 1 contains a printout of an image file of a photograph of a television displaying a recorded video of an actor. The video contains a superimposed eye in the corner, the logo of the network CBS. Most viewers are not tormented by this everpresent eye staring at them! In fact, most viewers are able to completely ignore the eye, and view the scene as though it didn't contain the stimulus, even if details such as the actor's sweatshirt's collar pass beneath the stimulus and are occluded by it.

Some stimulus is more everpresent than others. The Clay Mathematics Institute lists among its unsolved *Millennium Prize* problems the “red i removal problem.” This concerns the removal of stimulus (a red letter “i”) from images (Figure 2). The problem is particu-



Figure 1: Q. Who watches the TV watchers? A. CBS's all-seeing eye.

larly difficult because the information occluded by the i is completely gone, and because the authors of papers about the problem are persistently agitated because it seems like the letter should be capitalized.

In this paper I show how red i removal can be solved in certain specialized cases, using an artificial retinal network patterned after the brain contained within the human eye. Training this artificial retinal network is feasible on a single powerful desktop machine. Both training and execution of the model (a mere 400 megabytes) are GPU accelerated. The model presented in this paper was trained in about 3 days, and executing it in parallel on a suite of images takes about 100 milliseconds per image.¹

2 Artificial retinal networks

As I expertly foreshadowed in the previous section, an artificial retinal network works just like the brain inside a human eye. The retina is itself a rectangular 2D array of neurons, which turn photons into IEEE-754 floating point values between 0.0f and 1.0f. Behind this

*Copyright © 2015 the Regents of the Wikiplia Foundation. Appears in SIGBOVIK 2015 with the increasingly askance visage of the Association for Computational Heresy; *IEEEEEE!* press, Verlag-Verlag volume no. 0x40-2A. 0.00 Australian Neo-Dollars

¹Source code is available on the World Wide Web at: <http://sourceforge.net/p/tom7misc/svn/HEAD/tree/trunk/redi>

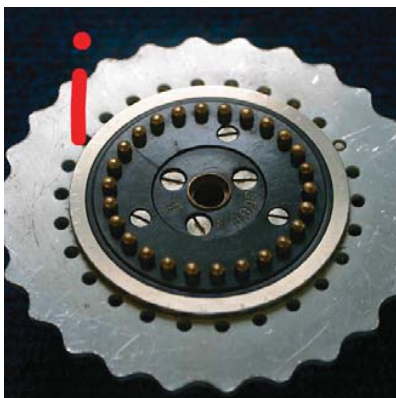


Figure 2: An image of an Enigma machine rotor with a red *i* superimposed. Solving this instance of the red *i* removal problem would mean producing an image without the red *i*. One way to do this would be to steal a floppy disk containing an original, unimposed image of the rotor, from someone in possession of it, for example the paper’s author.

is another 2D array of exactly the same size, except the pixels are in a weird jumbled order, and then another layer. This series of layers is known as the optic nerve. Finally, the brain perceives the image as an array of pixels, again of the same size (Figure 3).

It is easy to reconceptualize this process as an array of pixels undergoing several transformations. Obviously the story is more complicated: Humans see in color, so each pixel is actually three different nodes, one for red, green and blue. In fact, since some scientists hypothesize of certain “superseers”—that is, people who can perceive more than just the three wavelengths of light—we actually allow an arbitrary number of nodes per pixel. In this work, we used $N = 4$.

In a real human eye, each node is fed inputs from every node in the previous layer. For computational efficiency, in this work we allow only 64 inputs to each node from the previous layer. Because we suspect that layers are spatially related, a node is always connected to its *neighborhood* in the previous layer (each node from the 9 pixels within Manhattan distance 1). The rest of the inputs are selected randomly from a Gaussian distribution, as long as the samples fall within the image (using rejection sampling—the sides and corners do not “wrap around”). By the way, the images are always 256x256, because numbers that are a power of two are faster.² The connection from one node to another

²This is true on computers, because computers count in binary.

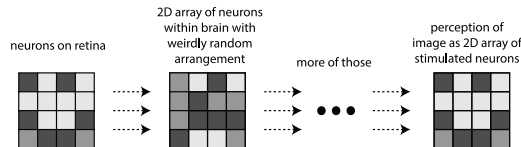


Figure 3: How eyes work, and thus, artificial retinal networks.

is modulated by a *weight*, again an IEEE-754 floating point number. A node outputs the sum of its input values, passed through a smoothulator, specifically the one found in Gray’s Anatomy (the book, not the TV show),

$$\frac{1}{1 + e^{-v}}$$

This function is *biologically plausible*.

We learn by backpropagated stochastic gradient descent, like babies do. Specifically: The network is presented with an image on its retina, and then we run the floating point values through the layers, summing them up and applying the smoothulator function, to produce a final image within the brain. Like a baby, it compares this image to what it expected to see, node by node. Where each node does not agree with the image, the error is computed. The baby propagates the partial derivative of the change in error with respect to the change in stimulus to the previous layer proportional to its weighted impact; fortunately the smoothulator has a simple derivative that is easily computed at a point from its output values. Error is not propagated into the real world (i.e., by sending light off of the retina back to the physical object that created the stimulus; that would be ridiculous).

2.1 Training data

One of the insights of this paper is that although artificial retinal networks require a lot of data to train, for certain problems the training data can be easily *generated*. For the red *i* removal problem, we begin with a corpus of about 4,000 images that were scraped from Google Image Search. Scraping images is easy; one just needs to list a bunch of queries for things that babies would want to view in order to learn what the world looks like. In this experiment I used terms such as [snakes], [dog on skateboard], [guitar], [stonehenge] and [superyacht]. Following this, I manually cleaned

In the human eye, powers of ten are faster, because humans have ten fingers.

the training data. I deleted images that were not photographic (drawings, etc.; for example, most images of guitars are actually 3D rendered cartoon guitars, fantasy images of guitars on fire, the Guitar Hero logo, and so on) or that were *too*, uh, pornographic (most queries for things that can have sex, like tapirs, contain prominent images of the things having sex). These are not appropriate for babies.

All images are cropped to a square and resized to 256x256. Then we generate training instances: An input image and the expected result. An image that does not contain a red *i* should just be transformed into the image itself (indeed, when we peer directly into the brain of a baby looking at a TV show, we find a region of the brain where the TV show is clearly visible). It is also easy to generate instances of the red *i* removal problem along with their solutions—we simply put a red *i* randomly on the source image and keep the destination image unchanged. In this way, we can easily generate a large amount of training instances (in actual practice, this procedure had a small bug; see Section 3.1). One unexpected phenomenon is that I had to be careful to remove images that already contained a red *i*, like many images of casinos, which are often called “CASINO”.

In order to coax networks into recognizing the *i*, we also place an *i* into the 4th color channel in the same position in the *expected output*. This is an invisible color channel which we discard, and which is always zero in the input. In essence we giving a hint to the eye’s brain that it should not just remove the red *i*, but it should also *perceive* it. I have not performed enough experiments to know if this is helpful.

For repeatability’s sake, important constants used in this experiment: There were 2 hidden layers. Gaussian samples were produced with a standard deviation of 16 pixels. The red *i* was rendered in Comic Sans, at a height of 80 pixels. I used a variety of learning rates, including an exponentially decreasing one (the standard advice of 0.05 is too large for constant learning rates on this kind of task, and limits the sharpness of resultant images).

2.2 CUDA, SHUDA, WUDA

Because we are working with graphical data, we should use the Graphics Processing Unit of the computer, not its Central Processing Unit (we are not processing centers). I implemented high-performance OpenCL kernels for each phase of training: The *forward* pass (signals flowing from the retina to the eye’s brain), the *back-propagation* step (when the eye computes the error and partial derivatives) and the *weight update* step (when

the eye rewires its neurons so that it sees the right thing next time). The phases have different parallelism constraints. Because the connectivity is sparse, we represent both forward and inverted index maps, which are decoded on the GPU. We take care to only load a single layer of the retinal network into the GPU’s RAM at once, to enable very large models, but we run many training instances in parallel for a single round. Some other operations, like the preparation of training data, are performed on the CPU. These are also frequently done in parallel, using C++11’s new `std::thread` with some crazy-ass wrappers to allow them to function in mingw32’s 64-bit gcc port. On a good day, training uses all 6 CPU cores and all 2800 GPU cores and about 14 GB of RAM and warms the home office like a 1kW space heater.

3 Results

After 4 rounds of training, the network produces an excellent-looking image that could be a *Cure* album cover, regardless of the stimulus cast upon its retina (Figure 4).

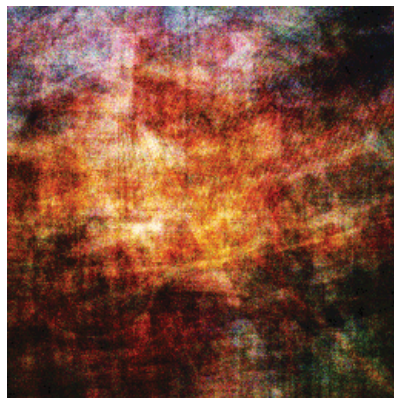


Figure 4: Result after 4 rounds of training. Looks great, and there is no red *i* to be seen, but it loses some points for not resembling the input image at all.

It’s not long before the network learns that it should not produce the same result for every input, and the output starts to mimic the input. These images look sort of like the world viewed through frosted glass (Figure 5), simulating how a baby first learns to see the world through the 1cm bulletproof Lexan of its translucent BabyLearn incubation cylinder.

Soon thereafter, the network begins to converge on

something like the identity function, as this drastically reduces error (even if some error is incurred by the persistence of the red i). Left overnight (about 9,000 rounds), we start to see the network both produce images much like the original (perhaps through a hip “vintage” Instagram filter), as well as removing the red i stimulus (Figure 6).

3.1 Evaluation

With a further 30,000 rounds of training, the output images sharpen and lose their Instagram quality (maybe only a small amount of “grain”), and the i is still successfully removed. However, since we’ve now made many passes over each image in the training set (and the model has about 100,000 degrees of freedom), it is certainly possible that we’ve simply overfit to this set of images (that is, that the baby’s eye’s brain is simply memorizing the i-free images and then recalling the one that looks closest to the stimulus). To evaluate fairly, we need to apply the model to totally new images that it was not trained on. These are called “eval” images.

Firing this up, I observed that the model successfully reproduced eval images that did not contain a red i; this is good because it means that it is not simply memorizing the training set images. I then started placing red i stimulus on the images with the mouse, and my heart sank: It wasn’t removing the red i at all! Dejected, I tried loading up the training images and putting a red i on them—it also did not remove the i, which did not make sense! Even for babies! Eventually, I discovered that the red i would be removed, as expected, but only when the i was in a handful of very specific locations. This was found to be a bug in the random i placing code; can you find it too?

```
uint8 x_dice = seed & 0x255;
seed >>= 8;
uint8 y_dice = seed & 0x255;
```

As a result, there are only 16 different possible x coordinates, and same for y . Nonetheless, this is still 256 different i locations that work, which implies considerable generality is possible. Due to Draconian SIGBOVIK deadlines, I have not yet been able to test a debugged training procedure.

Once the evaluation code only places an i at expected locations, the artificial retinal network works well (Figure 7)!

4 Conclusions

We find that the supposedly impossible red i removal problem is in fact solvable, at least in some forms, using artificial retinal networks. There are some limitations of the current model:

- It has only been tested to remove the letter i when it is rendered in bright red, in 30 point Comic Sans.
- It probably also removes letters like j , but maybe also in some other fonts, which is a sword that cuts both ways.
- Due to a bug, the red i must be at a position whose coordinates are exactly $\langle 12 + x_0 + 4x_1 + 16x_2 + 64x_3, 12 + y_0 + 4y_1 + 16y_2 + 63y_3 \rangle$, for x_j and y_j in $\{0, 1\}$.
- It automatically and non-optionally applies Instagram-style filters.

This technique can probably be applied to other image processing problems, for example, J peg dequantization. Here, we take an image and badly quantize it (for example, to 4 bits per color channel), and the training instance consists of the quantized image as input and the original image as the expected output; the retinal network learns how to fill in detail. Figure 8 shows the early stages (about 4000 rounds) of training such a model.

A related, still unsolved problem is “red i reduction”; here we do simply remove the i but replace it with a smaller i . For example, we could replace a capital I with a lowercase i , or replace a lowercase 30pt Comic Sans i with a lowercase 29pt Comic Sans i . This is an offshoot of the text ure compression field, which seeks to make the text “ure” smaller wherever it appears.

Biologically-inspired computer algorithms hold many wonders for those that seek to tap into the limitless potential of the 85% of the human eye’s brain that is currently unused. Perhaps humans even contain graphics processing units!

For higher-fidelity images and source code, please consult <http://tom7.org/redi>.

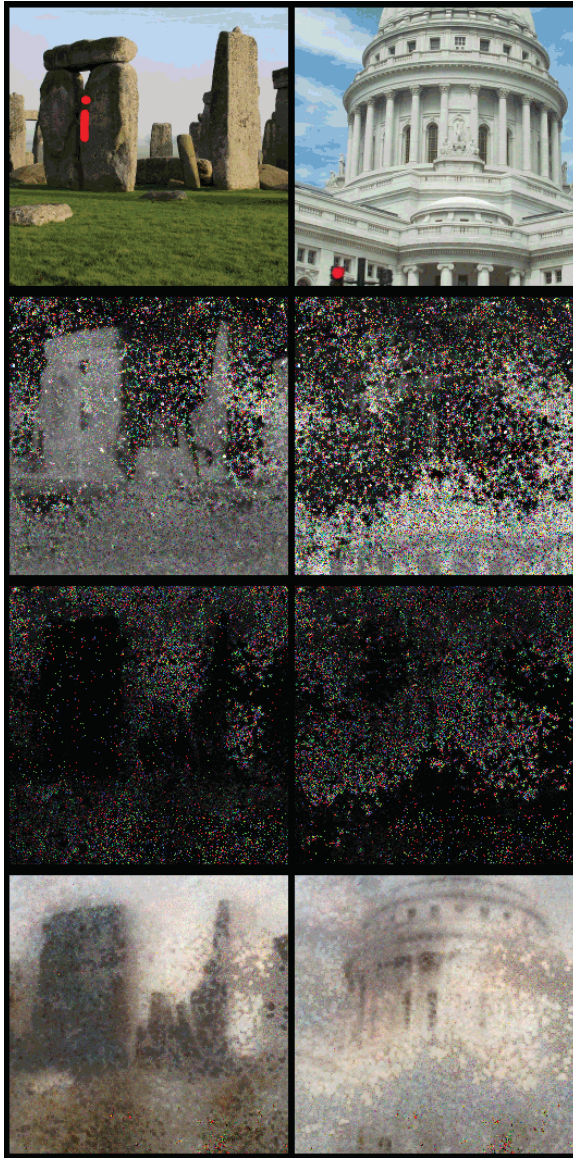


Figure 5: Result after 80 rounds of training, with the input image at the top and the signal proceeding downward through two hidden layers. The hidden layers aspire to crazy noise-terror glitch art versions of the stimulus as well.

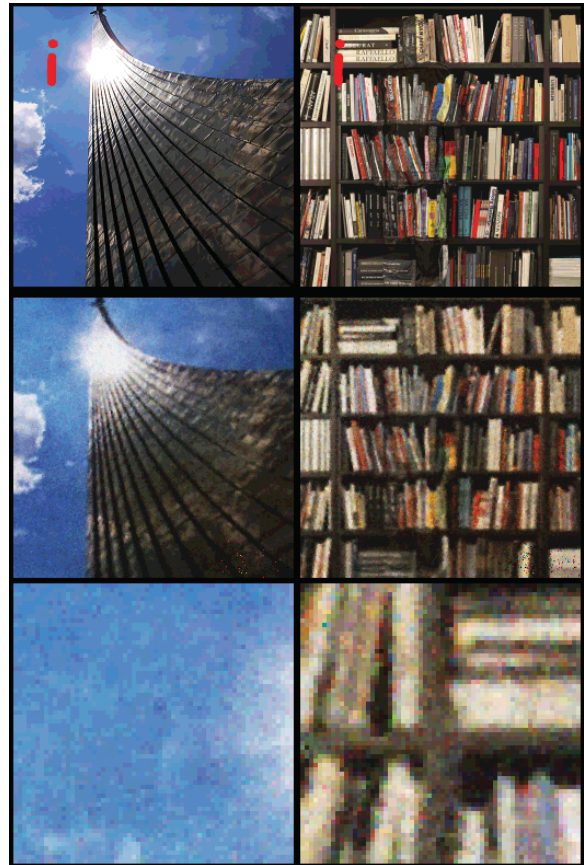


Figure 6: After about 9,000 rounds. Top row is the input image, the second row is the output (no longer showing hidden layers because they all just look like firefly raves); the bottom row shows 4x magnified detail of the region formerly containing the red i. Images are somewhat desaturated and blurry, but the red i is removed. Note how in the right image, the retinal network successfully continued both the horizontal and vertical bookshelves into the occluded region. This is not a trick.



Figure 7: Evaluation on new images after 30,000 rounds of training. Top row is the input image, the second row is the output, and third is 4x magnified detail. The first image (no i) shows the high amount of detail preserved. In the latter two, the i is successfully removed; the quality of the replacement is not perfect, but certainly reasonable.

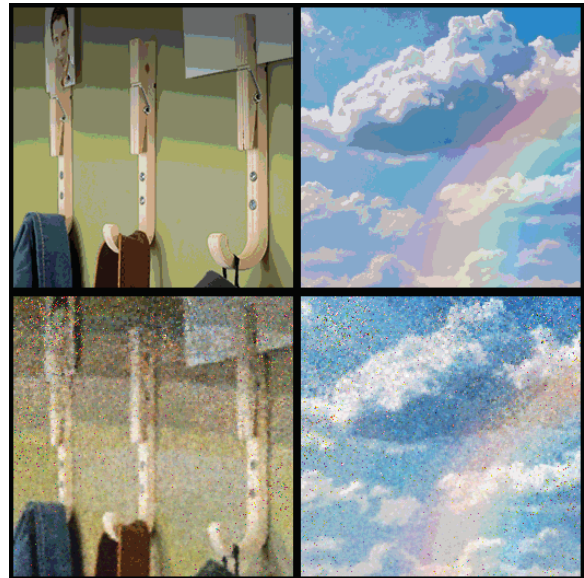


Figure 8: Evaluation of an early model for J peg dequantization. The model still contains a lot of noise pixels, which sometimes take a long time to converge, but it is already easy to see how quantization artifacts have been reduced (left). Actually, there is no reason why such dequantization must only be applied to J pegs; the right column shows it working on a nice rainbow picture.

A Proof of the Twelve Color Theorem: Hey, Nobody Bothered Before.

Sixth Grade Class Project for Mr. Blum

Y. T. Boveck Victoria V. Varg-Mack, VI

April 1 $\frac{1}{3}$, 2071

Editor's note: Several days ago, the wreckage of a small time machine appeared in the program committee's office, containing apparently only a copy of the proceedings from SIGBOVIK 2071. Unfortunately, all papers but one were burnt beyond recognition. Current speculation holds that the time machine operators forgot to disable the paradox safety interlock, and all the important (potentially causality-violating) papers were destroyed, leaving only this drivel. We're publishing it anyway, sorry.

Abstract

Roughly 100 years ago, mathematicians successfully proved the *4-Color Theorem*. Widely regarded as one of the most important theorems in graph theory at the time, the 4-Color Theorem states that for all planar graphs there exists an assignment of colors to nobes, such that no two adjacent nobes are assigned the same color, with at most 4 colors used. However, modern-day computers are capable of rendering far more diverse color palettes, and the old 4-color limitation is now largely irrelevant. In this work we extend the old result to support 12-colorings, offer some thoughts on generalization, and leave a conspicuous hole in our proof to support our future work.

1 Introduction

Victoria said I had to write this section, even though she's the only one of us who actually knows how the 4-color theorem proof goes. But luckily I found a proof for the 5-color theorem that made sense, and since having more colors is obviously better, that should work just as good. It goes like this:

Lemma 1.1. *All planar graphs have at least one nobe with at most 5 neighbors.*

Proof. Let $N(n)$ mean how many neighbors a given nobe n has. Then because each edge connects two nobes, the sum of all nobes' neighbors $\sum_n N(n) = 2e$. Let's assume all nobes have at least six neighbors. Then $6n \leq 2e$.

Next let $E(f)$ mean the number of edges that make up the border of a given face f of the graph. Since each edge borders 2 faces, $\sum_f E(f) = 2e$. Every region is bounded by at least 3 edges so that means $3f \leq 2e$.

If we plug all that into the "Euler's Forumula" that Mr. Blum gave us in class, which says that $n - e + f = 2$, here's what we get: $n - e + f \leq \frac{e}{3} - e + \frac{2e}{3} = 0$. But since $n - e + f = 2$ we end up with $2 \leq 0$. Then we must have been wrong to assume all nobes have at least six neighbors. That's a "contradiction"! \square

Theorem 1.2. *All planar graphs can be colored with no more than 5 colors so that no neighbor nobes have the same color.*

Proof. Let G mean the graph we're trying to prove this for. Using the lemon I just proved, pick the nobe n that has at most five neighbors and call those n_1, n_2, n_3, n_4, n_5 in whatever order you want, but remember the order! Let G' mean the smaller graph you get by taking out n , which of course is still planar. Because G' is smaller, we already know with "induction" that G' can be 5-colored. (That step seemed kinda weird to me but Victoria said it was legit so please don't take points off.)

Now we're gonna assume G can't be 5-colored. For this to be true, then each n_i from 1 to 5 must all have different colors, or else you could just pick the missing one for n . Let's think about the sub-graph that has only the 2 colors matching n_1 and n_3 , call it G_{13} . If the graph is disjoint, meaning n_1 and n_3 are in two separate parts, then you would be able to re-color the graph to make n_3 have color 1, so you can paint n with color 3. Therefore G_{13} must be connected. You can make the same argument for G_{24} to be connected, but suddenly that doesn't make any sense, because a path that keeps G_{24} connected would have to intersect the connected G_{13} !

So now, even if the 5-coloring of G' gives all different colors to n_1, n_2, n_3, n_4, n_5 , you should be able to re-color one of the n_i s so you can have a spare color to give to n . That means you can 5-color G . \square

Nice! Hey, that wasn't so bad.

2 The 12-Color Theorem

In the previous section, Y.T. et al. outlined the prior work in the field. Now we will make our main contribution; namely, to extend the 5-color theorem reviewed above to allow for 12-colorings of arbitrary planar graphs.

Theorem 2.1. *Given an arbitrary planar graph G , there exists an assignment of no more than 12 colors to the set of nobes $N(G)$ such that no two neighbouring nobes are assigned the same color.*

Proof. Using Theorem 1.2, generate a 5-color assignment for G . As 5 is less than 12, this assignment suffices. \square

Corollary 2.2. *If G has at least 12 nobes, then all 12 colors may be used.*

Proof. First we will prove the case for a smaller number of colors $c < 12$, by induction on c . Assume a graph G with c or more nobes, and an existing assignment of $c - 1$ colors in which all colors are represented, such that $c < 12$. The pigeonhole principle states that there must exist two nobes n_1 and n_2 with the same color. Simply assign the c th color to n_2 in the new coloring.

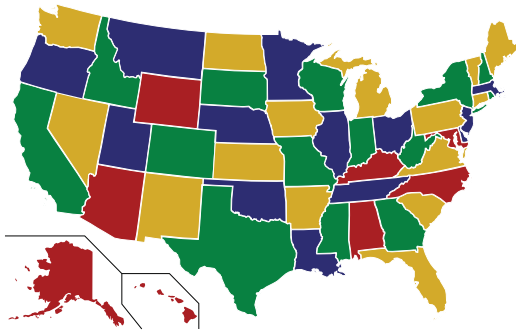
Next for the case $c = 12$. Given a graph G with 12 or more nobes, there must exist an assignment of 11 colors in which all colors are represented. The pigeonhole principle states that there must exist two nobes n_1 and n_2 with the same color. Assigning the 12th color to n_2 causes all 12 colors to be used. \square

An example application of our algorithm is depicted in Figure 1.

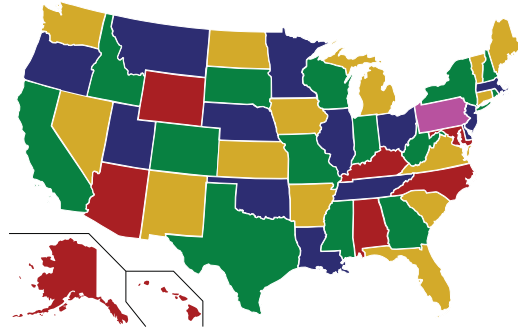
3 Conclusions and Future Work

In this paper we reviewed historical developments in the field, with a particular focus on the proof of the ~~4-Color Theorem~~ 5-Color Theorem; and we presented our main contribution, a constructive proof of our new *12-Color Theorem* based on the assumption of existing ~~4-colorings~~ 5-colorings of an arbitrary planar graph G .

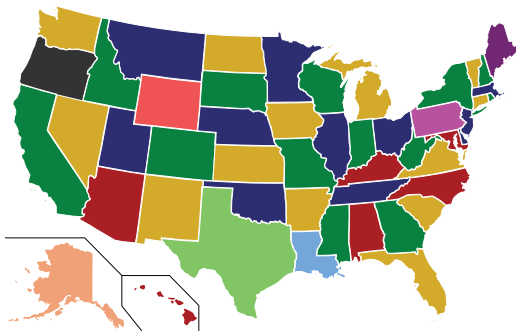
The most significant limitation of our work is that a generalized version of our proof only applies for C -colorings when $C \leq 12$. In future research, we plan on extending our theorem to support ever greater values of C . We hope the need for additional funding to study such advanced open problems is self-evident.



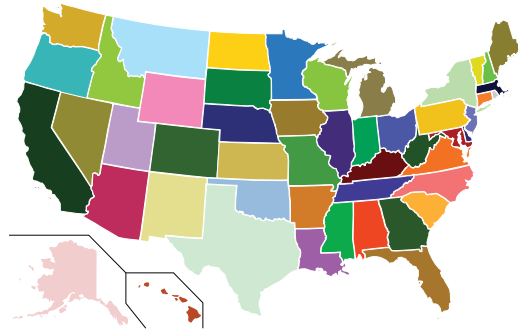
(a) Make a 4-color of your graph first.



(b) Choose a node and color it different.



(c) Repeat that seven more times to get a 12-coloring!



(d) If you had 51 different colors and wanted to use them all, it would look like this. However, we are leaving a proof of the existence of generalized 51-colorings to future work.

Figure 1: Mr. Blum said we needed to have some pictures in our paper, so we pretended the USA was a graph and colored it like our algorithm says.



CONFIDENTIAL COMMITTEE MATERIALS

SIGBOVIK 2015 Paper Review

Paper 10: A Proof of the Twelve-Color Theorem

Mr. Blum, New Pittsburgh Middle School

Rating: :(

Confidence: 4/4

This project makes a well-meaning, but ultimately entirely unconvincing, attempt at proving an interesting theorem in computer science. In the early twenty-first century, when the four-color theorem was still considered exciting, as opposed to something that all schoolchildren are taught from an early age, this might have been a reasonable starting point for sixth-grade work. Unfortunately, my standards are higher than this. I would have been more impressed if Y.T. and Victoria had started with a more recent and interesting result, such as the 2068 paper of Blum et al. (no relation) showing that $P = NP$.

Of course, as this class is “United States History: The 2017 Burrito Wars to the Present Day”, none of the above has any bearing. Using our school’s standard grading scheme of :) to :(, I assign this project a :(

Visually Identifying Rank

David F. Fouhey, *Mathematicians Hate Him!*
Daniel Maturana, *Random Forester Rufus von Woofles, Good Boy*

Abstract—The visual estimation of the rank of a matrix has eluded researchers across a myriad of disciplines many years. In this paper, we demonstrate the successful visual estimation of a matrix’s rank by treating it as a classification problem. When tested on a dataset of tens-of-thousands of colormapped matrices of varying ranks, we not only achieve state-of-the-art performance, but also distressingly high performance on an absolute basis.

Index Terms—perceptual organization; vitamin and rank deficiencies; egalitarianism in the positive-semi-definite cone; PAC bounds for SVDs; class-conscious norms

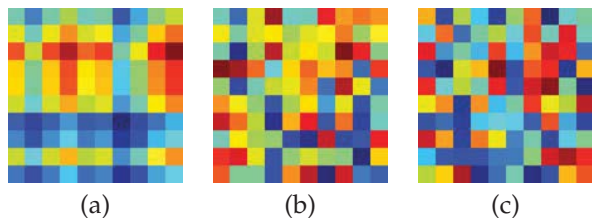


Fig. 1. What are the ranks of these matrices? Which ones are rank-deficient? In this paper, we investigate how one can guesstimate the rank of a matrix from visual features alone. See footnote on page 2 for answer.

1 INTRODUCTION

Consider Figure 1(b): what is the rank of the matrix? Most people are confused. Some might hazard a guess. A select collection of professors might say “3.” The mystery of how professors can visually estimate the rank of matrices from as little as a brief glance at a jet-colormap rendering has puzzled researchers in neuroscience, philosophy, mathematics, and computer science for decades.

The rank of a matrix M reveals a great deal. By definition, it tells us how many linearly independent columns the matrix has; surprisingly, it also tells us how many linearly independent rows the matrix has, and if that does not get you excited, I do not know what will. If we think of M as an operator, the rank tells us about the dimensionality of its output, and thus for a square matrix, whether M is invertible.

In this paper, we show how to identify the rank of a matrix from an image alone. In contrast to past work on guaranteed solutions to matrix rank computation

that require access to the matrix, our work gives guarantee-free solutions that can operate on only an colormapped version of a matrix. By treating matrix rank as an image classification problem, we are able to consistently achieve distressingly high performance – $\approx 40\%$ accuracy on 10-way classification; $\approx 80\%$ accuracy on rank-deficient/not-rank-deficient binary classification. In subsequent experiments we show the following: 1) Our method can identify what matrices seem low rank, and why; 2) Our method is easily extended to structured prediction; 3) That activations of our network can be even used as a feature for semantic image classification with non-embarrassing performance (20.9% on Caltech 101 with 15 samples).

2 RELATED WORK

In this work, we tackle two problems concerning a square matrix $M \in \mathbb{R}^{n \times n}$. The first is a binary classification problem: is M full-rank? The second is a k -way classification problem: what is the rank of M ?

Many approaches exist for solving both problems. In the binary case, for instance, note that a square matrix is full rank if and only if its determinant is non-zero. This leads to a straight-forward way to check for rank deficiency. Similarly, if we let $U\Sigma V^T = M$ be the singular value decomposition of M (i.e., $\Sigma_{i,i} = \sigma_i$ where σ_i is the i th singular value of M), then the rank is the number of non-zero singular values. This permits checking not only for rank deficiency but also calculating the rank.

While these sorts of approaches enable the accurate solution of both questions, they (a) require access to the matrix itself (as opposed to a screen capture or printout) and (b) have time complexity greater than $O(n^2)$. SVD computation has complexity $O(n^3)$ and determinant calculation is $O(n^3)$ with Bareiss [1] or $O(n^{2.807})$ with Bunch and Hopcroft [2]. Our work aims to fix these gaps.

Our method requires only access to a visual representation of the matrix, and thus answers the purely

• All authors are with The Robotics Institute, Carnegie Mellon University.
Send us fan mail at:
Neurotic Computing Institute c/o D. Fouhey, A.B. M.S.
EDSH 212
5000 Forbes Avenue
Pittsburgh, PA 15213

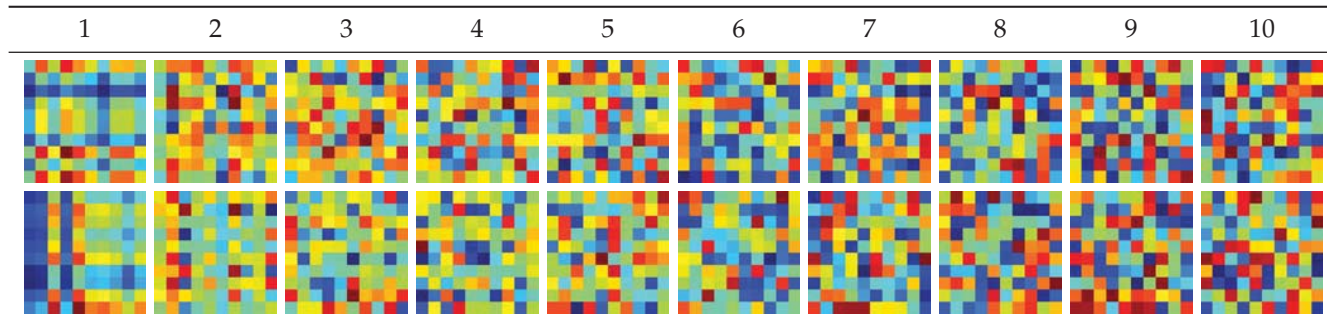


Fig. 2. Examples of matrices of various ranks. Top row: random instances; Bottom row: archtypical examples of each rank, determined by the most confident classification examples from a pool of 1,000 matrices according to a classifier.

visual way of computing rank as opposed to the *mechanical* way of computing rank. When a professor says a matrix looks rank-deficient, she is probably not doing an SVD, but instead using some visual smell-test (akin to the notion of direct perception as proposed by psychologist J.J. Gibson [3]); we seek to emulate this astounding ability.

Our method is $O(n^2)$. Feature extraction is only dependent on the number of pixels for all methods and thus $O(n^2)$. If we are doing binary classification, it is $O(1)$ and thus the method serves as a guarantee-free quadratic-time rank-deficiency test. If we do n -way classification, it depends on the method, but is arguably $O(1)$ for random forests, which most of our methods use.

We acknowledge that our work gives no guarantees, but computer vision has a long history of extraordinarily successful algorithms that may not always be right. The Random Sample Consensus algorithm [4], for instance, can give no guarantees about its performance on any individual problem instance. However, it works extraordinarily well in practice, and the authors have taken it all the way to the proverbial citation bank with a well-deserved $\approx 10,000$ citations. Similarly effective methods include Simulated Annealing, Iterative Conditional Modes (ICM) [5], and many others. Our method may not be as successful, but as presented in Table 1, our method is surprisingly effective and has a certain *je ne sais quoi*.

Our approach of neural-network based approaches to linear algebra is not itself novel. However, previous approaches (e.g., [6], [7], [8], among many others) require direct access to the matrix itself. Our approach, on the other hand, only gets access to an colormap of the matrix.

The natural complement to the thematically-related related work section (as above) is the alphabetically-related related work section introduced in Fouhey and Maturana’s seminal work on celebrity-themed learning [9]. In this paper, we extend this to a word-based

1. Answers first page quiz: a – rank 1; b – rank 3; c – rank 10.

TABLE 1

A comparison of ways to check whether a matrix is rank deficient. We evaluate methods on their time complexity, their success rate, whether there is an easy extension to n -way rank calculation, and their *Je Ne Sais Quoi*.

Method Name	Complexity (Time)	Success Rate	Multi-class Extension	Je ne sais quoi
SVD	$O(n^3)$	100%	Yes	Minimal
Det.	$O(n^{2.807})$	100%	No	Kinda
CNN	$O(n^2)$	78.6%	Yes	Tons

related work section. A highly related technique is the rank transform proposed by Zabih and Woodfill [10]. This method replaces each pixel by its “neighborhood rank” to achieve invariance to monotonic illumination differences. We employ a related technique, Local Binary Patterns [11] to extract features for our classifiers. Also related are learning-to-rank algorithms such as support vector ranking [12].

3 TECHNICAL APPROACH

We now introduce the method in simple English to illustrate its simplicity. We take a matrix, visualize it as a picture (like a .png), and feed it into a standard image classification pipeline. More formally, we create a fixed-length feature representation ϕ of the image, and learn a mapping f that maps the representation to a set of discrete classes. For instance, we might extract standard image features like SIFT as ϕ , and apply a standard technique like Random Forests or SVM to learn an f . Similarly, we might train a convolutional neural network (CNN) to predict the rank, serving as both ϕ and f . This classifier is simply trained on a collection of random matrices. We note that one elegant aspect of our method is that rank-deficiency and classification are encapsulated in the same learning formulation.

3.1 Features and Learning Method

In this paper, we apply standard image classification machinery by substituting in various standard fea-

TABLE 2

Quantitative results on visual rank problems. Our paradigm of rank-prediction works surprisingly well across a myriad of features and learning methods.

Learning method f	Random Forest+Engineered					RF + Pretrained		Scratch CNN	Chance
Feature map ϕ	<i>All Eng.</i>	<i>Gray SIFT BoW</i>	<i>Color SIFT BoW</i>	<i>LBP BoW</i>		<i>pool₅</i>	<i>fc₇</i>	<i>Raw Pixels</i>	
10-way Rank	38.1%	32.5%	36.5%	31.0%		33.7%	34.9%	43.5%	10%
Rank Deficiency	76.4%	73.1%	75.3%	73.9%		75.0%	76.3%	78.6%	50%

tures and learning methods for ϕ and f .

Shallow Learner + Features: *Features (ϕ):* The first feature type we use is standard hand-engineered features in the form of a bag-of-words (i.e., histogram) representation over dense SIFT [13] and Local Binary Patterns (LBP) [11]. To quantize SIFT, we build a codebook with k-means; each extracted SIFT feature is represented by the nearest cluster center (i.e., hard assignment). Thus, each image is mapped to one or more histograms of codewords; we concatenate histograms when using multiple representations. We also experiment with using the responses of a standard convolutional neural network – Alexnet [14] – pre-trained on the Imagenet dataset [15]. We use the standard $pool_5$ and fc_7 features.

Learning Methods (f): We work with random forests [16] although our method is entirely generic. In this method, an ensemble of decision trees is trained independently. During learning, splitting occurs on a random subset of features and occurs until a minimum number of samples is in a leaf. Whenever training, we do 5-fold cross-validation on the training data and select the values for both parameters (number of features considered, minimum node size) that give maximum mean performance.

Deep Learning: In keeping with the spirit of the deep learning times, we train a CNN to map directly from pixels to matrix rank. We refer to this as a Scratch CNN in the experiments since it is learned from scratch. Our experiments use a small amount of data, so we adapt a network designed for the MNIST dataset [17] that appears in the examples for [18]. Starting with all images resized to 60×60 , our network has architecture $C(5, 20) \rightarrow P(4, 3) \rightarrow C(5, 50) \rightarrow P(4, 3) \rightarrow C(4, 500) \rightarrow R \rightarrow \text{softmax}$, where $C(k, n)$ denotes a convolutional layer with n filters of size $k \times k$, $P(k, s)$ is max-pooling over a $k \times k$ region with stride s , and R is a rectified linear unit. Empirically, we found that the more aggressive max-pooling than usual helped the network generalize to matrices of other dimensions.

3.2 Implementation Details

We used Piotr Dollar’s toolbox [19], Vedaldi et al.’s VLFeat [20], MatConvNet [18], and LIBSVM [21]. *SIFT:* We extract and quantize SIFT on both the gray image and each of the R, G, and B channels separately; each codebook has 256 entries and one codebook is

generated on training data per channel. The codebooks are learned once on the 10×10 training set. *Scratch CNN:* We use a learning rate of 10^{-3} in a standard gradient-descent+momentum approach and 1M iterations; to prevent overfitting, we use the first iteration to have validation error within 1% of the final validation error.

4 EXPERIMENTS

We now rigorously evaluate our approaches for visually guesstimating rank-related matrix properties. *Every figure and table in this section represents a true experiment and actual results. We do not mess around.*

4.1 Dataset

We perform our experiments on a dataset of 10×10 matrices, with 2000 examples of each rank $1, \dots, 10$, which we split evenly into train and test. When doing binary rank-deficiency classification, we balance class distributions by downsampling the rank deficient class. We generate these matrices by first sampling a matrix M with entries uniformly and independently sampled from the interval $[0, 1]$. We then compute its SVD $M = U\Sigma V^T$ and set Σ to Σ but with the $r + 1, \dots, n$ th entries to 0 and compute $U\tilde{\Sigma}V^T$.

We convert each matrix to a 100×100 image, which we store as a PNG. This is done in MATLAB by calling the underlying colormapping functionality used by `imagesc` and then upsampling with nearest neighbor. In this paper, we primarily use the traditional and often criticized `jet` colormap, but we also experiment with two linear colormaps, `copper` and `bone`. All colormaps have 255 possible values and are scaled by the min and max of the matrices (i.e., the default of `imagesc`, where no absolute scale is imposed). Note that the many matrices of a variety of ranks may map to the same colormap visualization due to both colormap and PNG quantization.

4.2 Experiments – Features

In this section, we ask the question: what visual features are best suited for visually identifying the rank of a matrix? Is color a useful cue? It was argued in [22] that off-the-shelf pre-trained CNN features are an astoundingly effective baseline for any generic vision task – does this include profoundly unnatural

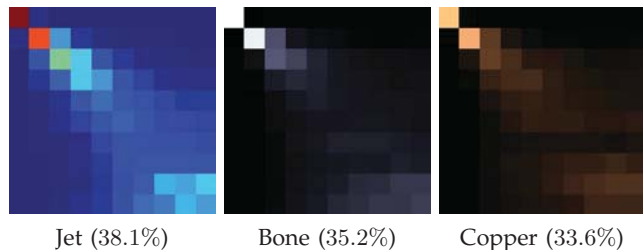


Fig. 3. Confusion matrices and accuracies for different colormaps on the 10-way rank classification problem using dense SIFT and LBP. Note that while there is variation, performance is decidedly above chance across colormaps.

images such as color-mapped matrices? Does learning a specialized CNN work on this task?

We present results in Table 2 showing the performance of various features. In the hand-engineered category, grayscale SIFT seems to perform on par with LBP; adding color considerably improves performance; and using all features does the best. The pretrained CNN does well despite the giant domain shift with both layers do slightly better than grayscale SIFT. However, in keeping with current results in computer vision, training a CNN from scratch consistently does the best. *Note, however, that all feature and method combinations operate at significantly above chance-level.*

4.3 Experiments – Colormaps

One natural question is whether the colormapping scheme affects the visual discrimination between matrices of different ranks. The *jet* colormap (e.g., Figs. 1, 5) in particular has received a lot of criticism for being difficult to interpret in practice by humans. Linear colormaps (i.e., smoothly varying from one color to another) in theory make for easier interpretation by humans. We see whether this holds true for computers as well. We compare the *jet* colormap (named for its origin in astrophysical fluid jet simulation) with *copper* and *bone*. The etymology of *copper* is – we hope for the reader’s sake – obvious; *bone* is so named because it looks somewhat like an X-ray and is popular because it lets researchers like us try to pretend to be brain-surgeons.

We run our learning method on matrices with a variety of colormaps and report 10-way classification results in Fig. 3 and 4. Generally, *jet* does the best. The only representation on which it does appreciably worse is the pre-trained CNN; we hypothesize this is because the linear colormaps produce more natural images, whereas the *jet* colormap’s outputs look like noise. Using grayscale SIFT, the results are roughly comparable, which is somewhat surprising as *jet* is known to convert poorly to grayscale. Nonetheless, while these differences exist, one consistent pattern is that the proposed method works surprisingly well across all colormaps.

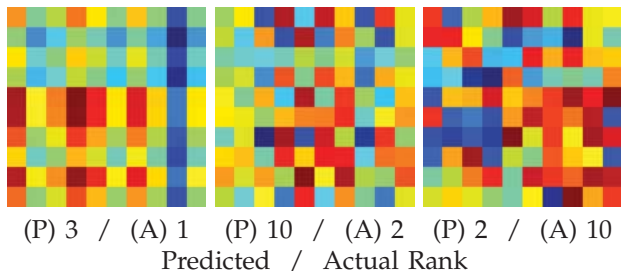


Fig. 5. Failure cases: some deceptive matrices with their (P)redicted and (A)ctual ranks, selected from the most confident mistakes of a RF classifier using dense SIFT and LBP features.

4.4 Experiments – Cross Domain

One recent pressing concern in the computer vision community is the biased nature of datasets: models learned on one dataset might not perform even reasonably on another, as reported in [23]. In our case, one might wonder whether a model learned to predict the rank of a 10×10 matrix (with a fixed set of ranks $1, \dots, 10$) can generalize to matrices of different sizes (e.g., 30×30). To answer this, we train a 10-class random forest on square matrices of dimensions 10, 15, and 30, and test them on different sizes; bag-of-words features are generated using the 10×10 matrix codebooks. These new matrix images have dimensions 150×150 and 300×300 respectively to maintain scale for the SIFT features. CNNs require a fixed input, and so we cannot apply this scaling trick to them.

We train a model to predict ranks $1, \dots, 10$ for all matrix sizes involved and report results in Table 3. Our method does surprisingly well, performing at around $2.5 \times$ chance-level when training on 10×10 matrices and testing on 30×30 matrices and vice-versa. The scratch CNN generalizes well, with the exception of the 30×30 scratch CNN on 10×10 data, which operates at chance level. This is poor generalization as opposed to a bad model to start with: the same model gets 49.7% when testing on 30×30 and 14.6% on 15×15 . We believe generalization could be improved by developing an architecture that would enable the row-to-pixel ratio to be constant.

5 DISCUSSION

The success of such a simple approach raises a number of questions, but our method also enables answer to some of these. For instance, we can see what makes a matrix smell rank deficient by analyzing the learned relationship between ϕ and f . We now discuss a few of these questions as well as extensions.

5.1 What does an archetypical rank- k matrix look like and which matrices are tricky to classify?

We can answer each of these questions by looking at the classifier scores; by looking at the most confident

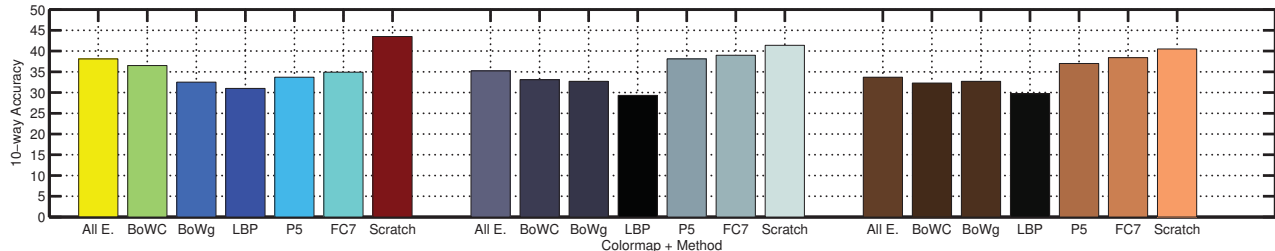


Fig. 4. What colormap is best for predicting matrix rank? (Left to right: jet, bone, copper). While Jet has been criticized widely compared to linear colormaps, it produces the best results with color sift and the from-scratch CNN

TABLE 3

Cross-domain performance: We report the accuracy of the methods on 10-way classification. Although chance on this task is 10%, most of our methods perform substantially better than chance.

Train Dim.	Test Dim.	Random Forest+Engineered				RF + Pretrained		Scratch CNN
		<i>All Eng.</i>	<i>Gray SIFT</i>	<i>Color SIFT</i>	<i>LBP</i>	<i>pool5</i>	<i>fc7</i>	<i>Raw Pixels</i>
10 × 10	10 × 10	38.1%	32.5%	36.5%	31.0%	33.7%	34.9%	43.5%
	15 × 15	33.7%	31.5%	33.5%	27.3%	27.3%	25.7%	37.9%
	30 × 30	25.9%	19.1%	25.3%	19.4%	17.7%	17.5%	24.1%
15 × 15	10 × 10	33.0%	28.1%	32.0%	26.5%	13.9%	14.4%	34.1%
		25.8%	22.7%	23.8%	21.6%	11.5%	11.8%	10.0%

mistakes of the classifier, we can find the most rank-deceptive matrices. We present some archetypical matrices in Fig. 2 according to RF classifier using all features. While the rank 1 matrix archetype is understandable, ranks 2 and up seem inscrutable. Nonetheless, the model is perfectly confident in its assessment of these matrices and is correct a surprising amount of time. Fig. 5 shows the model’s confident mistakes. On the left, for instance, is shown a rank-1 matrix with not too much apparent inter-row/column similarity that was mistakenly predicted to rank 3 by the RF.

5.2 What parts of matrices tell us rank?

Given our bag-of-words model, we can answer this by figuring out which codewords help the most in predicting rank as well as their sign (i.e., which codes are most associated with rank-1 rather than rank-10). We solve both by learning a L_2 -regularized logistic regression model to predict Rank- i or Rank- j , which we solve with LIBLINEAR [24]. The regularization parameter λ is selected via 5-fold cross-validation to give best average performance. The coefficients of the model w in terms of magnitude and sign indicate which codewords are indicative of rank deficiency.

We can visualize the informative regions of matrices by replacing pixels with the weight vector of their associated codewords. We show a few examples of this for low rank and high rank matrices using *grayscale* SIFT in Fig. 6. For rank 1, the regions associated with low rank have low frequency, and the codewords associated with high rank occur mainly at the sharp transition from the penultimate and blue column to the last column. The other ranks are a bit harder to

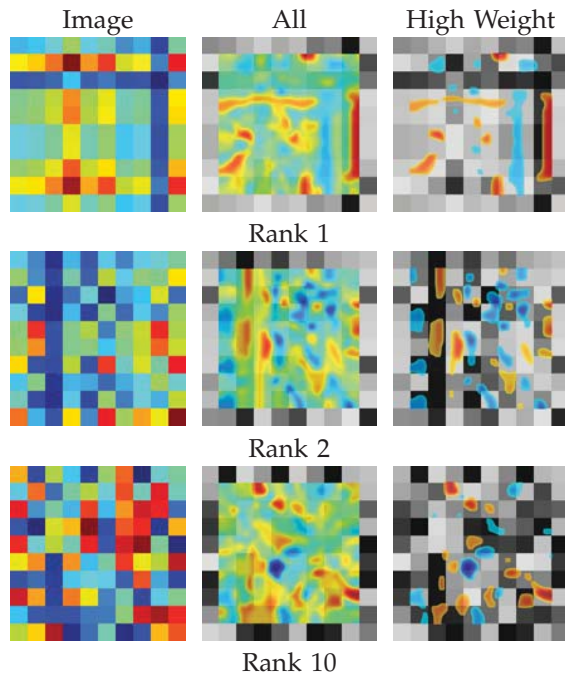














Fig. 6. A visualization of what makes a matrix look rank-deficient according to gray-scale SIFT. We train a logistic regressor to predict rank- k vs. full rank and plot weight-vector coefficients onto the image wherever the codeword appears. Blue is low rank, red high.

interpret, although the one can note the most strongly low-rank regions correspond to flat regions.

5.3 Can We Solve Structured Tasks?

So far, our approach of doing mathematics by learning has only been applied to classification problems. In-

Fig. 7. Examples from our deep visual multiplication net.

A	B	Pred. $A \cdot B$	True $A \cdot B$	MSE
				0.033
				0.014
				0.012

spired by our success in visual rank estimation, we are currently exploring the application of our framework to structured outputs, such as matrices.

In particular, we consider matrix multiplication and inversion, where for centuries mathematicians have relied on hand-crafted, shallow methods. Again, keeping with the times we propose to replace these methods with visual deep learning. To this end we designed a deep learning architecture, differing only in the input for each task: for the multiplication task we used two $M \times M$ concatenated multiplicands as input, whereas in the inversion task there is a single $M \times M$ input. The input is connected to two fully connected layers using ReLU nonlinearities of 512 hidden units each, followed by a fully connected $M \times M$ output layer with no nonlinearity. Dropout regularization was used in all layers. We generate $5 \cdot 10^5$ training examples for each task. In both cases we use 3×3 matrices with each entry independently sampled from a uniform(0,1) distribution as input, and their “true”² product and inverse as outputs.










We then use this data to train the network with stochastic gradient descent on a mean square error (MSE) loss for 100 epochs. Some qualitative predictions on unseen data are shown in Figures 7 and 8. We found the multiplication task to be easily solved by our network architecture, but the inversion task proved much more challenging, as shown by the higher MSE values. We note that this is analogous to humans taking Linear Algebra 101.

5.4 Can This Work On Real Data?

One advantage of our method is that it does not require access to the matrices themselves; but what if we only have a picture of the colormapped matrix? As a proof of concept, we took a cell-phone picture of each part of Fig. 1 of this paper, as shown in Fig. 9, left. We cropped out the matrix from the cell phone picture, as shown in Fig. 9, right. We then resized

2. At least according to our matrix library.

Fig. 8. Examples from our deep visual matrix inverse net.

A	Predicted A^{-1}	True A^{-1}	MSE
			0.25
			0.49
			1.56

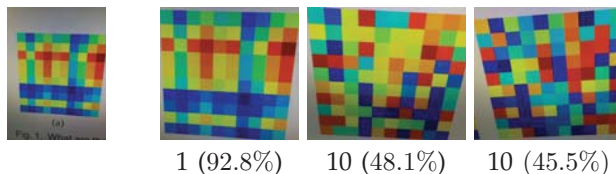


Fig. 9. Results on cropped images from cell-phone pictures of a computer monitor. (Left) sample pre-cropped image; (right) cropped images, their predicted rank, and posterior from the scratch CNN.

it and sent it through our scratch CNN. The rank 1 matrix was classified correctly, but both the rank 3 and rank 10 matrix were classified as rank 10. We note, however, that all images have perspective distortion that the CNN did not see at training time.

5.5 Does the matrix rank network generalize?

In our experiments, we confirmed the reports of [22] that one can use neuron activations from a network pretrained for classification as a strong feature for a variety of tasks; but can we use do the reverse? In other words, can we use activations in our scratch CNN as a feature for image classification?

To evaluate this, we tested our method on the Caltech 101 dataset [25]. We used the concatenated features from the last and second-to-last layers of the rank network (i.e., the softmax responses and the half-wave rectified feature map immediately before) as a feature representation. We then trained a multiclass SVMs (1v1, linear kernel) on top of these representations. We report results in Table 4; results are averaged over $1K$ random samplings of train sets; for test, we use an equal number per-class. While far from state-of-the-art, the numbers are respectable given that the underlying feature representation was trained to estimate matrix ranks.

6 CONCLUSIONS

In this paper, we introduced a new problem – visual rank estimation – and demonstrated that it is feasible

TABLE 4

Results on Caltech 101, training a linear SVM over responses from our scratch rank CNN. Chance on this dataset is $\approx 1\%$.

# Samples	5	10	15	20	25
Accuracy	12.6%	17.7%	20.9%	23.3%	25.1%

using conventional image classification approaches. Our approach is simple and obtains alarmingly high performance. More importantly, our features also convey understanding by showing us why some matrices just look low rank and what matrices have surprising rank. We have additionally demonstrated future directions in the form of structured prediction and have demonstrated that our rank predictor CNN can serve as a generic image feature.

REFERENCES

- [1] E. Bareiss, "Sylvester's identity and multistep integer-preserving Gaussian elimination," *Mathematics of Computation*, vol. 22, no. 102, 1968.
- [2] J. Bunch and J. Hopcroft, "Triangular factorization and inversion by fast matrix multiplication," *Mathematics of Computation*, vol. 28, no. 125, 1974.
- [3] J. Gibson, *The Ecological Approach to Visual Perception*. 1979.
- [4] M. Fischler and R. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. of the ACM*, vol. 24, June 1981.
- [5] J. Besag, "On the statistical analysis of dirty pictures," *Journal of the Royal Statistical Society, Series B (Methodological)*, vol. 48, no. 3, pp. 259–302, 1986.
- [6] J. J. Hopfield and D. W. Tank, "'Neural' computation of decisions in optimization problems," *Biological Cybernetics*, vol. 52, pp. 141–152, 1985.
- [7] M. Kennedy and L. Chua, "Neural networks for nonlinear programming," *Circuits and Systems, IEEE Transactions on*, vol. 35, pp. 554–562, May 1988.
- [8] A. Cichocki and R. Unbehauen, "Neural networks for solving systems of linear equations and related problems," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 39, pp. 124–138, Feb 1992.
- [9] D. F. Fouhey and D. Maturana, "The Kardashian Kernel," in *SIGBOVIK*, 2012.
- [10] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *ECCV*, 1994.
- [11] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *TPAMI*, vol. 24, no. 7, 2002.
- [12] T. Joachims, "Optimizing search engines using clickthrough data," in *KDD*, 2002.
- [13] D. Lowe, "Distinctive image features from scale-invariant keypoints," *IJCV*, vol. 60, no. 2, pp. 91–110, 2004.
- [14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.
- [15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," 2014.
- [16] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [18] A. Vedaldi and K. Lenc, "Matconvnet – convolutional neural networks for matlab," *CoRR*, vol. abs/1412.4564, 2014.
- [19] P. Dollár, "Piotr's Image and Video Matlab Toolbox (PMT)." <http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html>.
- [20] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms." <http://www.vlfeat.org/>, 2008.
- [21] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [22] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," *CoRR*, vol. abs/1403.6382, 2014.
- [23] A. Torralba and A. A. Efros, "Unbiased look at dataset bias," in *CVPR*, 2011.
- [24] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.
- [25] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," in *IEEE CVPR Workshop of Generative Model Based Vision (WGMBV)*, 2004.



David F. Fouhey David F. Fouhey received an A.B. from Middlebury College in Moose Watching in 2011. He is currently a Ph.D. student at the Robotics Institute at Carnegie Mellon University. He likes long walks, Edward Hopper, macchiatos, Jaffa Cakes, and, above all, kvetching. In his copious spare time, he sends fake announcements to the *New York Times'* Wedding Section. He and his colleagues were awarded the *People's Democratic Choice Award* at SIGBOVIK 2013.



Daniel Maturana Daniel comes from Chile. Daniel likes Eat'n Park, bicycling, recycling, and Pabst Blue Ribbon. You won't believe the one weird trick that credit card companies hate that Daniel uses to make money at home thanks to Obama lowering 10-year mortgage rates! He and his colleagues were awarded the *People's Democratic Choice Award* at SIGBOVIK 2013.



Rufus von Woofles Rufus von Woofles is a good boy, isn't he. Yesh he is. Rufus obtained his DoD (Doggy Obedience Diploma), First Class, from Muddy Paws University. Rufus is currently the PI of CHOCOLATE Lab at Carnegie Mellon University, where he leads research on predicting pizza-delivery-man appearance. Rufus likes wagging his tail and belly rubs. Rufus was awarded the *People's Democratic Choice Award* at SIGBOVIK 2013, but he tore it up.

Transparency Report

United States Three Letter Agency

1 April 2015

Here at Three Letter Agency, we care deeply about transparency. We have implemented a transparency policy carefully designed to ensure that transparency is at the heart of everything we do. Transparency means that you can set your mind at ease. Our track record proves that transparency is a top priority for us. We are more transparent than air itself. Transparency is our commitment to you. Our mission is to be the most transparent agency on the planet. We worry about transparency so that you don't have to. We take transparency very seriously. We use transparencies in all of our presentations. Finger lickin' transparent. Taste the transparency. Always transparent. We conduct regular transparency audits. We once briefly believed that we were not being transparent, but it was a mistake. A transparent proxy intercepts normal communication at the network layer without requiring any special client configuration, such that clients need not be aware of the existence of the proxy. Transparency keeps you safe. We have a long tradition of transparent processes and outcomes. We make transparency work for you. Transparency is in our DNA. Think transparent. Got transparency? So transparent, no wonder it's number one. Every breath you take, every move you make, we'll be transparent.

Track L

Lunch Intermission

1. Burritos for the Hungry Mathematician

Ed Morehouse

Keywords: burritos, monads, why did i spend my day doing this instead of real work?

Burritos for the Hungry Mathematician

Ed Morehouse

April 1, 2015

Abstract

The advent of fast-casual Mexican-style dining establishments, such as Chipotle and Qdoba, has greatly improved the productivity of research mathematicians and theoretical computer scientists in recent years. Still, many experience confusion upon encountering burritos for the first time.

Numerous burrito tutorials (of varying quality) are to be found on the Internet. Some describe a burrito as the image of a crêpe under the action of the new-world functor. But such characterizations merely serve to reindex the confusion contravariantly. Others insist that the only way to really understand burritos is to eat many different kinds of burrito, until the common underlying concept becomes apparent.

It has been recently remarked by Yorgey [9] that a burrito can be regarded as an instance of a universally-understood concept, namely, that of monad. It is this characterization that we intend to explicate here. To wit,

a burrito is just a strong monad in the symmetric monoidal category of food; what's the big deal?!

1 The Category of Food

The **category of food**, $\mathbf{F\ddot{U}D}$, is a full subcategory of the category of stuff, \mathbf{STF} , whose objects are the stuff you can eat. A morphism of this category is known as a **recipe**. This category inherits the ambient *monoidal product*: if you can eat a carrot and you can eat a potato, then you can eat a carrot and a potato. The *monoidal unit* in the category of food is nothing, ε , which is to be found in many graduate students' refrigerators, typically, tensored with beer.

The monoidal category $\mathbf{F\ddot{U}D}$ is *symmetric*: if you can eat a carrot and a potato then you can eat a potato and a carrot. However, it fails to be *cartesian*. We lack the projections: if we have made the mistake of ordering a whiskey sour, there is generally no way to recover just the whiskey. Nor do we have a diagonal natural transformation, $\Delta(A) : A \rightarrow A \otimes A$, giving us two whiskeys for the price of one – alas. For further details on the theory of categorical cookery, see [1].

2 The Tortilla Endofunctor

The category of food supports an endofunctor, $T : \text{FÜD} \rightarrow \text{FÜD}$, known as the **tortilla endofunctor**. Objects in the image of this functor are precisely those that are wrapped in a tortilla. The action of the functor T on morphisms is to take a recipe and yield a recipe from tortilla-wrapped ingredients to tortilla-wrapped results.

Note that the tortilla endofunctor is, in general, *not* monoidal. Given tortilla-wrapped rice and beans, it is usually not possible to recover tortilla-wrapped rice and tortilla-wrapped beans. Nor is it always possible to produce a tortilla-wrapped nothing out of nothing.

3 The Burrito Monad

It is universally understood that a **monad** on a category \mathbb{C} is comprised of [6] an endofunctor $T : \mathbb{C} \rightarrow \mathbb{C}$ and two natural transformations, $\eta : \text{id}(\mathbb{C}) \rightarrow T$ and $\mu : T^2 \rightarrow T$, known respectively as the monad **unit** and **multiplication**, satisfying the monoid unit and associative laws:

$$(\eta \cdot T) \cdot \mu = \text{id}(T) = (T \cdot \eta) \cdot \mu : T \rightarrow T$$

and

$$(\mu \cdot T) \cdot \mu = (T \cdot \mu) \cdot \mu : T^3 \rightarrow T$$

In the case of the burrito monad, the functor in question is, of course, the tortilla endofunctor.

The burrito monad unit takes something you can eat and wraps it in a tortilla, yielding a **simple burrito**, which is both tortilla-wrapped and edible – no mean feat. This constitutes the familiar *insertion of generators* [5]. However, in the case of less-viscous generators, such as guacamole, some care may be required in the insertion, and it is often easier to insert the generator into the monad using a nozzle rather than by the obvious folding map.

The burrito monad multiplication takes a **compound burrito**, that is, something you can eat that is double-wrapped in tortillas, and merges the two tortillas into a single wrapping. This tends to happen spontaneously in the case of an overabundance of the previously-mentioned less-viscous generators, although a similar effect can be achieved mechanically by using some force and the *flattening* function (should we cite the crappy Gabor paper here?) [2].

Now we must check that the monad laws are satisfied:

monad right unit law: This law says that if you begin with something you can eat that's wrapped in a tortilla and you wrap the whole thing in another tortilla, then merge the tortillas, you get back what you started with, as the reader may deliciously verify.

monad left unit law: This law says that if you begin with something you can eat that's wrapped in a tortilla and you unwrap it, remove the contents

and wrap them in another tortilla, then wrap the result back up in the original tortilla and merge the two wrappings, the effect is again the identity function.

monad associative law: This law says that if you have a triple-tortilla-wrapped thing you can eat and you merge the inner two tortillas, then merge the result with the outer tortilla, the effect is the same as if you had merged the outer two tortillas first, and then the result with the inner one.

4 From Burritos, Strength

The burrito monad is in fact strong. A monad (T, η, μ) on a monoidal category (\mathbb{C}, \otimes, I) is **strong** if there is a natural transformation,

$$\tau(A, B) : A \otimes T(B) \longrightarrow T(A \otimes B)$$

satisfying certain relations [3]. In the case of a *strict* monoidal category, such as $\mathbf{F\ddot{u}D}$, these amount to the requirement that τ be a *morphism* of the monad structure. The corresponding laws are:

$$\text{id}(A) \otimes \eta(B) \cdot \tau(A, B) = \eta(A \otimes B) : A \otimes B \longrightarrow T(A \otimes B)$$

and

$$\text{id}(A) \otimes \mu(B) \cdot \tau(A, B) = \tau(A, T(B)) \cdot T(\tau(A, B)) \cdot \mu(A \otimes B) : A \otimes T^2(B) \longrightarrow T(A \otimes B)$$

In the case of the burrito monad, the strength, τ , is the “wrapping in” transformation, which takes a “side” object A and a burrito $T(B)$ and yields a burrito in which the side has been wrapped in to the burrito. This is most clearly illustrated by an example. Suppose that you begin with a bean burrito and a side of guacamole. By “wrapping in” the side, you may obtain a bean and guacamole burrito – which is clearly better. In this case, the monad strength laws require the following:

strength unit law: If you first insert some beans into a simple burrito and next wrap-in a side of guac, the effect is the same as if you had inserted the guac and beans into a simple burrito together.

strength multiplication law: If you have a compound bean burrito that you first merge into a simple one and you then wrap-in a side of guac, the effect is the same as if you had wrapped the guac into the outer, bean burrito, burrito, then wrapped the (now interstitial) guac into the inner, bean, burrito, and then finally merged the compound guac and bean burrito

into a simple one. This last equivalence is perhaps best understood as:

$$\begin{array}{lcl}
 & & \text{guac and ((bean burrito) burrito)} \\
 \Rightarrow & \text{guac and ((bean burrito) burrito)} & \Rightarrow \text{[wrap-in guac to outer burrito]} \\
 & \text{[merge compound burrito]} & \text{(guac and (bean burrito)) burrito} \\
 & \text{guac and (bean burrito)} & \Rightarrow \text{[wrap-in guac to inner burrito]} \\
 \Rightarrow & \text{[wrap-in guac]} & \text{((guac and bean) burrito) burrito} \\
 & \text{(guac and bean) burrito} & \Rightarrow \text{[merge compound burrito]} \\
 & & \text{(guac and bean) burrito}
 \end{array}$$

5 Functional Burritos

Those coming to burritos from a functional programming background may prefer to understand them by way of the “return and bind” formulation of monads [8], popularized by the *typeclass* mechanism of languages such as Haskell.

From this perspective, the burrito monad’s `return` is just the same as its unit η . Flipping argument order, we see that the `bind` operation, $(A \rightarrow TB) \rightarrow TA \rightarrow TB$, transforms a burrito recipe into one that expects to receive its ingredients in burrito form. The monad laws in this formulation are satisfied just in case one takes `flip bind f` to be $T(f) \cdot \mu(B)$.

The T-image of the morphism f performs f while wrapped in a tortilla. Depending on the nature of the morphism, this may raise size issues. In the preceding discussion, we have assumed all burritos to be locally small.

6 Conclusion and Related Work

As you can see, burritos are not so hard to understand, once viewed from the proper perspective – namely, that of abstract category theory. We hope that this little tutorial will encourage more researchers to incorporate burrito-theoretic results and methods into their own work.

Burrito-like sandwiches have emerged several times independently, but were not put on a solid theoretical footing until their connection to monads became apparent in the early 1970s [3, 7]. More recently, a connection to the *Hegelian taco*, a display of the 3-dimensional eight-element graphic monoid with five left ideals, has been worked out by Lawvere [4]. It is conjectured that weak equivalences exist to other wrap-like structures, but research in this area is presently ongoing.

References

- [1] Bob Coecke. “Quantum Pictorialism”. In: *Contemporary Physics* (2009). URL: <http://arxiv.org/abs/0908.1787>.

- [2] Cat Ferguson. *Overly honest references: “Should we cite the crappy Gabor paper here?”* URL: <http://retractionwatch.com/2014/11/11/overly-honest-references-should-we-cite-the-crappy-gabor-paper-here/>.
- [3] Anders Kock. “Strong Functors and Monoidal Monads”. In: *Archiv der Mathematik* 23 (1972), pp. 113–120.
- [4] F. William Lawvere. “Display of Graphics and their Applications, as Exemplified by 2-Categories and the Hegelian Taco”. In: *Proceedings of the First International Conference on Algebraic Methodology and Software Technology*. 1989.
- [5] Saunders Mac Lane. *Categories for the Working Mathematician*. second edition. Graduate Texts in Mathematics. Springer, 1998.
- [6] Mark Molloy. *Grammar crusader spends years removing repeated error 47,000 times on Wikipedia*. URL: <http://www.telegraph.co.uk/men/the-filter/11392756/Grammar-crusader-spends-years-removing-repeated-error-47000-times-on-Wikipedia.html>.
- [7] Ross Street. “The Formal Theory of Monads”. In: *Journal of Pure and Applied Algebra* 2 (1972), pp. 149–168.
- [8] Philip Wadler. “Monads for Functional Programming”. In: *Advanced Functional Programming*. Lecture Notes in Computer Science 925. Springer-Verlag, 1995.
- [9] Brent Yorgey. *Abstraction, intuition, and the “monad tutorial fallacy”*. URL: <https://byorgey.wordpress.com/2009/01/12/abstraction-intuition-and-the-m Monad-tutorial-fallacy/>.



CONFIDENTIAL COMMITTEE MATERIALS

SIGBOVIK 2015 Paper Review

Paper 7: Burritos for the Hungry Mathematician

A. G.

Rating: 3 (weak accept)

Confidence: 2/4

This paper presents a novel category theoretic understanding of burritos. This understanding is a major step forward in the eventual goal of understanding the categorical structure of *Füd*. The definition of the tortilla endofunctor is a significant research contribution that is a start at illuminating the mysterious structure of *Füd*. However, the reviewer is not satisfied with just the definition of the tortilla endofunctor and believes that either the author should define the other endofunctors in this category or prove a uniqueness theorem as to why the tortilla endofunctor is the only one possible. This reviewer suspects that there may actually be infinite endofunctors in *Füd* once one considers, crêpes, blini, wraps, etc – the whole set of circular two dimensional vessels for enclosing food.

Track Z

Computationalizing Computation

1. A New Paradigm for Certified Code

Stefan Muller

Keywords: certified code, certified compiler, proof-carrying code

2. Beyond the Halting Problem: Higher Order Infinite Loop Checkers

Jody Leonard and Aaron Santiago

Keywords: computability, decidability, infinite, loop, checker

3. Bashing Haskell:

Reimplementing the Parsec Library Inside the Unix Shell

Mike Izbicki

Keywords: unix, sed, parsing, parsec, haskell, applicative, monad

A New Paradigm for Certified Code

Stefan Muller

Carnegie Mellon University
smuller@cs.cmu.edu

Abstract

Everyone likes running programs, but before you run a piece of code, you want to be sure that it's actually code. Utterly ignoring building on a large body of work in *certified code*, we solve this ever-so-common problem.

1. Introduction

Dozens of prior studies over several years have worked in the area of “certified code” (e.g. [1–5, 7, 9, 10, 12–14]). Like all great researchers, the present author didn't read any of these papers, as such a literature study clouds one's mind with lesser ideas and stifles true innovation. Instead, this paper presents an entirely novel way of generating and running certified code. You're welcome.

In these fast-and-loose days of computing, it is common to assume that a given file is an executable binary, run `chmod +x` on it, and execute it willy-nilly. However, doing this on a non-executable file can severely damage one's computer¹. It is thus vital to the security of computers that, prior to executing a file, we *certify* that it is, in fact, *code* (as opposed to, for example, a humorous animated GIF of a cat). This is the domain of *certified code*. In this paper, we describe CODECERT, a new, language-independent method of generating certified code from source, and running it safely.

2. The CODECERT System

The key observation underlying CODECERT is that executable binaries are the result of running a compiler on a piece of source code. This observation, explained in Figure 1, means that, to certify that a file is a binary, and therefore code, we need only certify that it is the output of a compiler.

¹ Or gracefully throw an exception and exit, as the case may be.

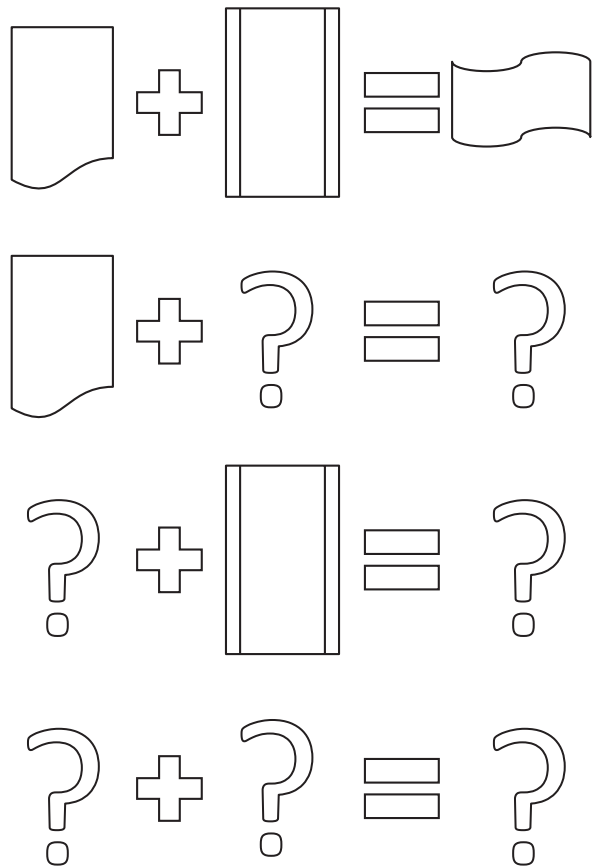


Figure 1: A binary is the output of a compiler on source code. If the inputs are not a compiler and source code, no guarantees are possible.

Permission to make shallow or deep copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage (because that would just be mean), that copies bear this notice and that bears copy this notice.

SIGBOVIK '15, April 1, 2015, Pittsburgh, Pennsylvania, USA.
Copyright © 2015 ACH ... \$1000.00 (or best offer)

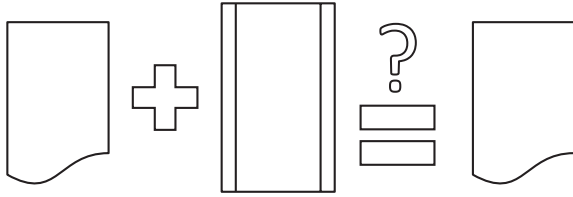


Figure 2: The CODECERT system compiles a given source file with a given compiler and compares the result with a given file. The file is a binary if and only if the results match.

Compilers	Not compilers
gcc	ls
javac	awk
ghc	echo
ocamlc	sleep
gc	apt-get
coqc	emacs
fbc	bash

Table 1: Comparison of compilers and non-compilers

The simplest way to do this is to keep as proof artifacts the source code and the compiler, and show that the purported binary is actually what is obtained by running the purported compiler on the purported source. This process is explained in further detail in Figure 2.

To produce certified code, the CODECERT system takes a purported binary, along with its source code and the compiler with which it was compiled. These three files are packaged together. When the certified binary is to be run, CODECERT compiles the packaged source code with the packaged compiler. If this matches the purported binary, the binary is executed. Otherwise, the file is determined to not be a binary, and the user is warned not to execute it.

2.1 Certified Compilers

At this point, the astute reader may observe that the process described above works only if the compiler provided to CODECERT is actually a compiler. Otherwise, the “source code” could be, for example, an image and the “compiler” could be a program that, for example, adds captions to images. This concern led us to conduct novel research in the area of *certified compilers* [6, 8], related to certified code. Our method for certifying that a program is a compiler relies on a second key observation. This observation was reached after detailed study of a large number of compilers and non-compiler utilities, catalogued in Table 1. Note that all compilers in the list end in the letter “c” while no non-compilers do. The CODECERT code generator takes advantage of this fact to certify that the provided binary is a compiler before generating the package.

```

$ ocamlc -o hello hello.ml
$ ./mkcode ocamlc hello.ml hello hello_pkg
$ ./runcode hello_pkg
Hello, world!
$ ./mkcode ocamlc hello.ml cat.jpg hello_pkg
$ ./runcode hello_pkg
Code not certified!

```

Figure 3: In the first instance, CODECERT correctly determines that `hello` is code. In the second, it correctly determines that `cat.jpg` is not code.

3. Proof of Correctness

We now formally prove that CODECERT is correct in that running a certified code package generated by CODECERT will proceed only if the provided file is indeed code.

Theorem 1. *If CODECERT produces a certified code package from a file e , then either e is an executable or running the code package using CODECERT will produce an error.*

Proof. By induction on the length of the filename of e . A zero-character filename is invalid, so the base case is trivial. If the length is n , rename e to a name of $n - 1$ characters. By induction, this new file, and therefore e , is either an executable or will produce an error. \square

4. Implementation

Our CODECERT package consists of two utilities: `mkcode` and `runcode`. The `mkcode` utility takes as command-line arguments a source code file, its compiler, the name of the uncertified binary and the name of the certified package to output and, as the name suggests, mks a certified code package. This utility is 12 lines of Bash script code which certifies the compiler and, if valid, prepares a `.tar` archive of the provided files, along with the command line with which to invoke the compiler².

The second utility, `runcode`, extracts the files from the archive, invokes the compiler and uses `diff` to compare the output with the provided binary. If they match, the binary is executed. Otherwise, an error is produced. This utility consists of nine lines of Bash code. Figure 3 shows two interactions with CODECERT.

5. Related Work

A closely related avenue of research which we hope to soon explore (read: solve) is *proof-carrying code* [11]. For example, see Figure 4.

References

- [1] A. W. Appel and A. P. Felty. A semantic model of types and machine instructions for proof-carrying code.

²This is assumed to be `./<compiler> -o <output> <src>`. If the compiler arguments do not take this form, `mkcode` will fail gracefully and output an error. We assume. This hasn’t actually been tested.

```

1 (* Suppose  $\sqrt{2} = a/b$  where  $a, b$  are not both even.
2 *  $2b^2 = a^2$ , so  $a^2$  is even, so  $a=2n$ 
3 *  $2b^2 = 4n^2$ 
4 *  $b^2 = 2n^2$ 
5 *  $b^2$  is even, so  $b$  is even. Contradiction.
6 *)
7
8 let rec fib n =
9   if n <= 1 then 1 else (fib (n - 1)) + (fib (n - 2))

```

Figure 4: This code carries a proof that $\sqrt{2}$ is irrational. It also computes Fibonacci numbers.

In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '00, pages 243–253, New York, NY, USA, 2000. ACM. ISBN 1-58113-125-9. doi: 10.1145/325694.325727. URL <http://doi.acm.org/10.1145/325694.325727>.

- [2] S. Chaki, J. Ivers, P. Lee, K. Wallnau, and N. Zeilberger. Model-driven construction of certified binaries. In *Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems, MODELS'07*, pages 666–681, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-75208-0, 978-3-540-75208-0. URL <http://dl.acm.org/citation.cfm?id=2394101.2394161>.
- [3] K. Crary and S. Sarkar. Foundational certified code in the twelf metalogical framework. *ACM Trans. Comput. Logic*, 9(3):16:1–16:26, June 2008. ISSN 1529-3785. doi: 10.1145/1352582.1352584. URL <http://doi.acm.org/10.1145/1352582.1352584>.
- [4] K. Crary and J. C. Vanderwaart. An expressive, scalable type theory for certified code. In *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming, ICFP '02*, pages 191–205, New York, NY, USA, 2002. ACM. ISBN 1-58113-487-8. doi: 10.1145/581478.581497. URL <http://doi.acm.org/10.1145/581478.581497>.
- [5] N. A. Hamid, Z. Shao, V. Trifonov, S. Monnier, and Z. Ni. A syntactic approach to foundational proof-carrying code. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science, LICS '02*, pages 89–100, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1483-9. URL <http://dl.acm.org/citation.cfm?id=645683.664592>.
- [6] X. Leroy. Formal certification of a compiler back-end or: Programming a compiler with a proof assistant. In *Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '06, pages 42–54, New York, NY, USA, 2006. ACM. ISBN 1-59593-027-2. doi: 10.1145/1111037.1111042. URL <http://doi.acm.org/10.1145/1111037.1111042>.
- [7] T. Lindholm and F. Yellin. *Java Virtual Machine Specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1999. ISBN 0201432943.
- [8] J. S. Moore. A mechanically verified language implementation. *J. Autom. Reason.*, 5(4): 461–492, Nov. 1989. ISSN 0168-7433. URL <http://dl.acm.org/citation.cfm?id=83471.83477>.
- [9] G. Morrisett, K. Crary, N. Glew, D. Grossman, R. Samuels, F. Smith, D. Walker, S. Weirich, and S. Zdancewic. Talx86: A realistic typed assembly language. In *In Second Workshop on Compiler Support for System Software*, pages 25–35, 1999.
- [10] T. Murphy, VII. Ml grid programming with concert. In *Proceedings of the 2006 Workshop on ML, ML '06*, pages 2–11, New York, NY, USA, 2006. ACM. ISBN 1-59593-483-9. doi: 10.1145/1159876.1159879. URL <http://doi.acm.org/10.1145/1159876.1159879>.
- [11] G. C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '97, pages 106–119, New York, NY, USA, 1997. ACM. ISBN 0-89791-853-3. doi: 10.1145/263699.263712. URL <http://doi.acm.org/10.1145/263699.263712>.
- [12] G. C. Necula. *Compiling with Proofs*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1998. AAI9918593.
- [13] Z. Shao, B. Saha, V. Trifonov, and N. Papaspyrou. A type system for certified binaries. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '02, pages 217–232, New York, NY, USA, 2002. ACM. ISBN 1-58113-450-9. doi: 10.1145/503272.503293. URL <http://doi.acm.org/10.1145/503272.503293>.
- [14] Z. Shao, V. Trifonov, B. Saha, and N. Papaspyrou. A type system for certified binaries. *ACM Trans. Program. Lang. Syst.*, 27(1):1–45, Jan. 2005. ISSN 0164-0925. doi: 10.1145/1053468.1053469. URL <http://doi.acm.org/10.1145/1053468.1053469>.

Beyond the Halting Problem

Higher Order Infinite Loop Checkers

Jody Leonard and Aaron Santiago

jleonard11@simons-rock.edu asantiago11@simons-rock.edu
Division of Science, Mathematics and Computing, Bard College at Simon's Rock

Abstract. The infamous *Halting Problem* asks whether it is possible to determine whether an arbitrary program P halts on arbitrary input x . It is well known that the Halting Problem is undecidable for all input pairs of P and x - however, this question nevertheless remains woefully unexplored. In this landmark paper, we reframe the investigation with a more practical construction: *infinite loop checkers*.

Keywords: computability, decidability, infinite, loop, checker.

1 Introduction

In 1936, Alan Turing's delivered his paper "On Computable Numbers, with an Application to the Entscheidungsproblem" [3] as a response to the famous *Entscheidungsproblem*¹. In essence, Turing showed that for all programs P and inputs to those programs x , the language

$$L_H := \{(P, x) | P \text{ halts when run on } x\},$$

also known as the *Halting Problem*, is not computable on a Turing Machine. Since the Halting Problem can be reduced to the Entscheidungsproblem, it follows that the Entscheidungsproblem has no general solution.

Our paper does not challenge Turing's conclusions, but rather draws attention to some important concerns and limitations of Turing's circumstances:

- Before the 1990s, there was no sufficient method to test any of Turing's results because programming didn't exist

¹ Elvish for "tree fecal issue". For more information, see [2].

yet. The release of Python and other programming languages have all but solved this problem.

- Technology now has reached levels of computational power that would be unfathomable to researchers of Turing's time. The clock speed of Turing Machines are lumps of dirt compared to the chips found in even the light switches of today.
- Alan Turing was a much too attractive man. This meant that his ability to design theoretically valid statements must have been impaired by his ability to socialize.

With these limitations in mind, it is important to reassess the foundations of Turing's arguments in order to continue computing as a whole. Even though no one can know for sure, the authors believe that without such a change in direction, computing will be forced to stop.

To this end, we re-define and operationalize the Halting Problem in terms of a construction we call the *infinite loop checker*. Critically, this construction allows us to better consider extrapolations of the Halting Problem, which we term *orders* of infinite loop checkers.

Lemmas 1, 2, and 3 lead us to consider Theorem 1.

Theorem 1. *If n is of the form $n = 2k$, then ILC_n is decidable; if n is of the form $n = 2k + 1$, then ILC_n is undecidable.*

To test Theorem 1, we created an arbitrary order infinite loop checker generator, and verified the first few trillion orders of infinite loop checkers. Pseudocode is given in Algorithm 2, and Table 2 outlines the fruits of our research up through ILC_{50} . So far, the results of this test have been consistent with Theorem 1. We leave the formal proof of Theorem 1 as an exercise to the reader⁶.

Algorithm 2 Infinite Loop Checker Generator

Input: n , a Natural number.

Output: P , a program that is an implementation of ILC_n ; False if not possible.

```

if  $n$  is greater than 0 and even then
    return "return False"
else
    return False
end if

```

5 Infinite Order Infinite Loop Checkers

Building on Definition 3, we can begin to approach the concept of high-order ILC_n programs theoretically. First, we observe a new approach to Definition 3 in Remark 1

Remark 1. ILC_n for $n \geq 1$ is a program that verifies its input to be ILC_{n-1} .

To cover the case of ILC_1 , we provide Definition 4.

Definition 4. ILC_0 is an infinite loop.

⁶ Any completed exercises should be submitted to coders@simons-rock.edu for review.

Table 2: Infinite Loop Checkers

ILC Order	Decidable?	Runtime
ILC_1	False	N/A
ILC_2	True	$O(1)$
ILC_3	False	N/A
ILC_4	True	$O(1)$
ILC_5	False	N/A
ILC_6	True	$O(1)$
ILC_7	False	N/A
ILC_8	True	$O(1)$
ILC_9	False	N/A
ILC_{10}	True	$O(1)$
ILC_{11}	False	N/A
ILC_{12}	True	$O(1)$
ILC_{13}	False	N/A
ILC_{14}	True	$O(1)$
ILC_{15}	False	N/A
ILC_{16}	True	$O(1)$
ILC_{17}	False	N/A
ILC_{18}	True	$O(1)$
ILC_{19}	False	N/A
ILC_{20}	True	$O(1)$
ILC_{21}	False	N/A
ILC_{22}	True	$O(1)$
ILC_{23}	False	N/A
ILC_{24}	True	$O(1)$
ILC_{25}	False	N/A
ILC_{26}	True	$O(1)$
ILC_{27}	False	N/A
ILC_{28}	True	$O(1)$
ILC_{29}	False	N/A
ILC_{30}	True	$O(1)$
ILC_{31}	False	N/A
ILC_{32}	True	$O(1)$
ILC_{33}	False	N/A
ILC_{34}	True	$O(1)$
ILC_{35}	False	N/A
ILC_{36}	True	$O(1)$
ILC_{37}	False	N/A
ILC_{38}	True	$O(1)$
ILC_{39}	False	N/A
ILC_{40}	True	$O(1)$
ILC_{41}	False	N/A
ILC_{42}	True	$O(1)$
ILC_{43}	False	N/A
ILC_{44}	True	$O(1)$
ILC_{45}	False	N/A
ILC_{46}	True	$O(1)$
ILC_{47}	False	N/A
ILC_{48}	True	$O(1)$
ILC_{49}	False	N/A
ILC_{50}	True	$O(1)$

Pseudocode for an implementation of ILC_0 is provided in Algorithm 3, and is pending runtime analysis.

Algorithm 3 Infinite Loop

```

while True do
  Nothing.
end while

```

Now consider ILC_∞ . Remark 1 states that

$$ILC_\infty \text{ verifies } ILC_{\infty-1},$$

but $\infty - 1 = \infty$, which implies that

$$ILC_\infty \text{ verifies } ILC_\infty$$

or simply, that an *infinite order infinite loop checker* is a program that verifies itself. Pseudocode for an implementation of an infinite order infinite loop checker is provided in Algorithm 4.

Algorithm 4 Infinite Order Infinite Loop Checker

```

Input: P a program.
Output: True if P is equivalent to this program; False otherwise.
Generate own source code S
if P = S then
  return True
else
  return False
end if

```

6 Additional Results and Open Problems

Using the research techniques found in Sections 2, 3, and 4, we consider Definition 5 and suggest Lemmas 4 and 5 as a cursory exploration of the possibility of infinite order infinite loop checker checkers. We offer these Lemmas as open problems.

Definition 5.

$$IOILCC_n \begin{cases} \text{verifies } IOILCC_{n-1} & \text{if } n > 0 \\ \text{is } ILC_\infty & \text{if } n = 0 \end{cases}$$

Lemma 4. $IOILCC_1$ is not computable.

Lemma 5. $IOILCC_2$ is computable.

The authors would like to explore the possibility of higher order infinite order infinite loop checker checkers themselves, however their machines are currently still stuck analyzing the runtime of the infinite loop algorithm, and it would seem that emulating C64 hardware on a five year old laptop is not granting them any favors.

Also, the possibility of exploring checkers in multiple dimensions might divulge more insight on the nature of Turing's original halting problem.

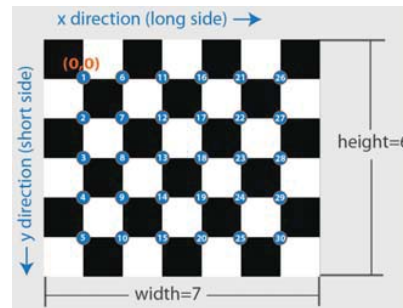


Fig. 1: A possible explanation of multi-dimensional infinite loop checkers.

References

1. D. J. Rumsey. *Statistics For Dummies*. For Dummies, 2nd edition, 2011.
2. Naomi Saphra. The dumping lemma: Assessing regularity. *SIGBOVIK 2014*, pages 51–52, April 2014.
3. A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

Bashing Haskell: Reimplementing the Parsec Library Inside the Unix Shell

Functional Imperative Pearl

Mike Izbicki

University of California Riverside
mike@izbicki.me

Abstract

We introduce the `Parsed` suite of Unix command line tools. `Parsed` improves the classic `sed` program by incorporating ideas from the popular `parsec` Haskell library. In particular, the Unix pipe operator `(|)` corresponds exactly to the Applicative bind `(*>)` in Haskell. The resulting syntax is both intuitive and powerful. The original syntax used by `sed` can match only regular languages; but our improved syntax can match any context sensitive language.

Keywords unix, sed, parsing, parsec, Haskell, applicative, monad

1. Introduction

The stream editor (`sed`) is one of the oldest and most widely used Unix utilities. Unfortunately, it is a monolithic beast. It fails to live up to the Unix philosophy of “do one thing well.” The problem is that `sed` tries to implement all regular expression features in a single executable. This ruins composibility. For example, let’s say I have a simple `sed` command for deleting email addresses. Something like:

```
sed 's/[a-zA-Z0-9.][a-zA-Z0-9.]/xxx@xxx/g'
```

But in the file I’m working on, I only want to delete an email address if the line starts with the words `CONFIDENTIAL`. This should not be a hard task. We *should* be able to write another `sed` command for finding lines that begin with `CONFIDENTIAL`, then combine these two commands. But this is not possible using standard techniques. To solve our problem, we must rewrite an entirely new `sed` command from scratch. Within that command, we copy-and-paste our previously tested command:

```
sed 's/(CONFIDENTIAL.*)[a-zA-Z0-9.][a-zA-Z0-9.]/$1xxx@xxx/g'
```

Real world experience suggests that this practice is a leading source of bugs for the modern `sed` programmer.¹ In this paper, we simplify shell based parsing by introducing techniques from functional programming into the Unix shell. Our `Parsed` library combines the best of `sed` with Haskell’s excellent `Parsec` library.

`Parsec` is a simple Haskell library that makes parsing easy and fun. It provides a small set of simple higher order functions called combinators. By combining these combinators, we can create programs capable of parsing complex grammars. The `Parsed` library recreates these combinators within the Unix terminal. In particular, each combinator corresponds to either a shell script or shell function. We then combine these combinators using the standard Unix pipe. There are three combinators of particular importance: `match`, `some`, and `choice`. With these combinators, we can match any regular expression. But unlike `sed`, our combinators can

¹ Can you spot the bug in the command?

take advantage of the shell’s built-in expressivity to match any context sensitive language!

In the remainder of this paper, we give a brief tutorial on how to construct your own parsers using `Parsed`. We recommend that you install `Parsed` and follow along with our examples. Installing is easy. Just clone the git repo:

```
$ git clone https://github.com/mikeizbicki/parsed
And add the resulting parsed/scripts folder to your PATH:
```

```
$ export PATH="$ (pwd) /parsed/scripts:$PATH"
```

That’s it! You’re now ready to start parsing!

2. Matching strings

The most basic combinator is `match`. It’s easiest to see how it works by example. Throughout this tutorial, we will first present the examples, and then their explanation.

```
$ match hello <<< "goodbye"
$ echo $?
1
```

`match` takes a single command line argument, which is the string our parser is trying to match. In the above example, this is the string `hello`. The text to be parsed comes from `stdin`. In the above example, we use bash’s built-in `<<<` syntax, which redirects the contents of the following string (`goodbye`) to `stdin`. The exit code of the previously run program is stored in the shell variable `$?` . In the above example, our parser failed (the string `hello` does not match the string `goodbye`), so `$?` contains a non-zero value.

Now let’s see what a successful parse looks like:

```
$ match hello <<< "hello"
hello
$ echo $?
0
```

When `match` succeeds, the matched text gets printed to `stderr`. This is where the output string `hello` comes from in the above example. Since parsing succeeded, `match` returned an exit code of 0. As you can see, `match` is a very simple parser. It is normally combined with other parsers.

3. Combining parsers

We combine parsers using the standard unix pipe `(|)`. For example, let’s create a parser that first matches the string `hello` and then matches the string `world`:

```
$ (match hello | match world) <<< "helloworld"
helloworld
$ echo $?
0
```

Script 1 match

```
#!/bin/sh

# check for the correct number of arguments
if [ -z "$1" ] || [ ! -z "$2" ]; then
    echo "match_requires_exactly_one_argument"
    exit 255
fi

# When reading from stdin, the shell ignores the
# contents of the IFS variable. By default,
# this is set to whitespace. In order to be
# able to read whitespace, we must set IFS to
# nothing.
IFS=''

# read in exactly as some characters as are in
# the first command line argument
read -rn "${#1}" in

# if we parse correctly
if [ "$1" = "$in" ]; then
    # print the parsed string to stderr
    echo -n "$in" 1>&2

    # forward the unparsed contents of stdin
    cat

    # signal that parsing succeeded
    exit 0
else
    # parsing failed
    exit 1
fi
```

Parsing succeeds for both calls to `match`, so parsing succeeds overall. To see how piping combines parsers, we need to take a more careful look at the `match` script. Like all combinators in the Parsed library, `match` is written in POSIX compliant shell. The full source code is shown in Script 1 above.

We've already seen that when `match` succeeds, the matched string is printed to `stderr`; but additionally, any unparsed text is printed to `stdout`. This is demonstrated by the following two tests:

```
$ match hello <<< "helloworld" 1> /dev/null
hello
$ match hello <<< "helloworld" 2> /dev/null
world
```

As a reminder, by putting a number in front of the output redirection operator `>`, we redirect the contents of that file descriptor to the specified file. File descriptor 1 corresponds to `stdout` and 2 to `stderr`. So the first command above discards `stdout`; it shows only the text that was successfully parsed. The second command above discards `stderr`; it shows only the text that is sent to any subsequent parsers.

It is possible for the first parser to succeed and the second to fail. In this case, the succeeding parsers still print their output to `stderr`, but parsing fails overall:

```
$ (match hello | match world) <<< "hellogoodbye"
hello
$ echo $?
1
```

Concatenating two `match` parsers is relatively uninteresting. We could have just concatenated the arguments to `match`! So now let's introduce a new combinator called `eof` which matches the end of file. That is, `eof` will succeed only if the `stdin` buffer has been

Script 2 eof

```
#!/bin/sh

# Try to read a single character into the
# variable $next. If next is empty, we're at
# the end of file, so parsing succeeds.
IFS=
read -n 1 next
if [ -z $next ]; then exit 0; else exit 1; fi
```

closed because there is no more input to parse. The source code for `eof` is much simpler than for `match`, and is shown in Script 2.

These next two examples demonstrate the effect of the `eof` combinator. First, we try matching the string `hello` on the input `hellohello`:

```
$ match hello <<< "hellohello"
hellohello
$ echo $?
0
```

Parsing succeeds; we print the contents of the parsed text (`hello`) to `stderr`; and we print the remaining content to be parsed (`hello`) to `stdout`. If, however, we apply the `eof` combinator:

```
$ (match hello | eof) <<< "hellohello"
hello
$ echo $?
1
```

Parsing now fails because the input was too long. Shortening the input again causes parsing to succeed:

```
$ (match hello | eof) <<< "hello"
hello
$ echo $?
0
```

Now that we know how to combine two simple parsers, we're ready for some more complex parsers.

4. Iterating some parsers

The `some` combinator lets us apply a parser zero or more times. `some` corresponds to the Kleene star (`*`) operator used in `sed` and most other regular expression tools. `some` takes a single command line argument, which is the parser we will be applying zero or more times. For example:

```
$ (some "match_hello" | eof) <<< "hellohello"
hellohello
$ echo $?
0
```

The above example succeeds because the `some "match hello"` parser consumes *both* occurrences of `hello`. As already mentioned, `some` need not consume any input:

```
$ some "match_hello" <<< ""
$ echo $?
0
$ some "match_hello" <<< "goodbye"
$ echo $?
0
```

In fact, the `some` used by itself can never fail—it will always just parse the empty string. In order to force failures, we must concatenate `some` with another parser like so:

```
$ (some "match_hello" | eof) <<< "goodbye"
$ echo $?
1
```

Script 3 some

```
#!/bin/sh

# check for the correct number of arguments
if [ -z "$1" ] || [ ! -z "$2" ]; then
    echo "many_requires_exactly_one_argument"
    exit 255
fi

# put the contents of stdin into a variable so we
# can check if it's empty
stdin=$(cat)

# if we still have input and parsing succeeded
if [ ! -z "$stdin" ] && stdout=`eval "$1" <<< "$stdin"`; then

    # run this parser again
    "$0" "$1" <<< "$stdout"
else

    # stop running this parser
    cat <<< "$stdin"
fi
```

Unfortunately, the `some` combinator breaks the ability to use POSIX pipes for concatenation as we did above. Consider this example:

```
$ (match hello | some "match_hello") <<< ""
$ echo $?
0
```

Our first `match` parser failed because there was no input. Then we call the `some` parser. There is still no input, so `some` succeeds. Since `some` was the last command to execute, `$?` contains its exit code which is 0.

The problem is that the standard POSIX `$?` variable has the wrong behavior. We need a variable that will report if any of the commands in the pipe chain failed. There is no POSIX compliant way to do this. But more modern shells like `bash` offer a simple fix. The command

```
$ set -o pipefail
```

modifies the semantics of the `$?` variable so that it contains zero only if all commands in the pipe chain succeed; otherwise, it contains the exit code of the first process to exit with non-zero status. This command need only be run once per `bash` session. Thereafter, we can rerun the incorrect example above to get the correct output:

```
$ (match hello | some "match_hello") <<< ""
$ echo $?
1
```

The code for the `some` combinator is shown in Script 3 above.

5. Custom combinatorial explosion

Most regular expression libraries offer a primitive combinator + that matches one or more occurrences of a previous parser. `Parsec` does not have such a primitive operator because it is easy to build it using only the `|` and `some` combinators. We call the resulting operator `many`, and we implement it by creating a bash function:

```
$ many() { $1 | some "$1"; }
```

Recall that the `$1` variable will contain the first parameter to the `many` function. In this case, that parameter must be a parser. We

Script 4 many

```
#!/bin/bash
set -o pipefail
$1 | some "$1"
exit $?

$ many "match_hello" <<< ""
$ echo $?
1
$ many "match_hello" <<< "hello"
hello
$ echo $?
0
```

can use our new combinator to simplify the examples from the previous section:

Unfortunately, `Bash` functions do not last between sessions. If we want something more permanent, we can create a script file instead. In fact, the `many` combinator is so useful that it comes built-in to the `Parsec` library. You can find its source code in Script 4 above. When the script file starts, it will not inherit the settings of its parent process in the same way that our `many` function did. Therefore, we must specifically re-setup our non-POSIX pipes at the beginning of every script that uses them.

6. The program of choice

The last combinator we need for parsing regular languages is called `choice`. The `match` and `many` combinators required exactly one input, but `choice` takes an arbitrary number of input parameters. Each parameter is a parser. For each of these parsers, `choice` applies it to `stdin`. If it succeeds, then `choice` returns success. If it fails, then `choice` goes on to the next parameter. If all parsers fail, then `choice` fails as well.

Here's a simple example using just the `match` combinator:

```
$ choice "match_hello" "match_hola" <<< hello
hello
$ echo $?
0
$ choice "match_hello" "match_hola" <<< hola
hola
$ echo $?
0
$ choice "match_hello" "match_hola" <<< goodbye
$ echo $?
1
```

Our parser succeeds when the input was either `hello` or `hola`, but fails on any other input.

7. Free your context

In the intro we claimed that `Parsec` is strictly more powerful than `sed`. We now demonstrate this point by parsing a context free language. `Sed` only supports regular expressions. The reason `Parsec` has this extra power is because we can define our own recursive parsers using standard shell syntax.

To demonstrate this capability, we will write a parser that checks for balanced parenthesis. First, let's define a small combinator `paren` that takes a single parser as an argument and creates a parser that succeeds only when the parameter is surrounded by parentheses:

```
$ paren() { match "(" | $1 | match " "; }
```

And let's test it:

Script 5 choice

```
#!/bin/bash

# We need to store the contents of stdin
# explicitly to a file. When we call one of
# our candidate parsers, it will consume some
# of the input from stdin. We need to restore
# that input before calling the next parser.
stdin=$(tempfile)
cat >"$stdin"

# If stdin is empty, then we're done with
# recursion; parsing succeeded
if [ ! -s "$stdin" ]; then
    exit 0
fi

# For similar reasons, we'll need to store the
# output of our parsers.
stdout=$(tempfile)
stderr=$(tempfile)

# For each parser passed in as a command line arg
for cmd in "$@"; do

    # Run the parser; if it succeeds, then pass
    # on its results
    if $(eval "$cmd" <"$stdin" 1>"$stdout" 2>"
        $stderr") ; then
        cat "$stderr" >&2
        cat "$stdout"
        exit 0
    fi
done

# All parsers failed, so we failed
exit 1
```

```
$ paren "match_hello" <<< "hello"
$ echo $?
1
$ paren "match_hello" <<< "(hello)"
(hello)
$ echo $?
0
```

We will use `paren` to write a combinator `maybeparen` that accepts whether or not the parameter parser is surrounded by parenthesis. Here is a reasonable first attempt:

```
$ maybeparen() { choice "$1" "paren_$1"; }
```

Unfortunately, this doesn't work due to scoping issues with `bash`. When we run our command, we get an error:

```
$ maybeparen "match_a" <<< "(a)"
choice: line 7: paren: command not found
```

We get this error because the `choice` combinator is its own shell script. When this script gets executed, a new shell process starts with a clean set of environment variables. In the context of this subprocess, the `paren` function we created above doesn't exist.

There are two ways to solve this problem. The simplest is to use the following `bash`-specific syntax:

```
$ export -f paren
```

This command tells the running `bash` shell that any subshells it spawns should also have access to the `paren` function. Now when we try running our previous tests:

Script 6 parens

```
#!/bin/bash
choice "$1" "match_" ('_|_par_\ "$1\_|_match_')' "

$ maybeparen "match_a" <<< "a"
a
$ echo $?
0
$ maybeparen "match_a" <<< "(a)"
$ echo $?
1
```

We still get a parse error! What's happening is that we actually defined the `maybeparen` function incorrectly. Here is the correct definition:

```
$ maybeparen() { choice "$1" "paren_\ "$1\""; }
```

The only difference is that the correct definition surrounds our `$1` parameter with quotation marks. This ensures that the entire value of the variable gets passed as a single parameter to the `paren` combinator. Because these quotation marks are within quotation marks, they must be escaped with backslashes. Arrgh!

We are now ready to define a combinator `parens` that matches any number of balanced parentheses. In order to avoid the need for a set of triply nested quotation marks, we will put the `parens` combinator within a script file (instead of a function) and we will manually inline our call to the `paren` combinator. Script 6 above contains the final result. And now let's test our creation:

```
$ par "match_a" <<< "(((a)))"
(((a)))
$ echo $?
0
$ par "match_a" <<< "((b))"
$ echo $?
1
$ par "match_a" <<< "(((a"
$ echo $?
1
```

We've successfully parsed a context free language :)

8. Discussion

This is only a very brief introduction to the capabilities of our `Parsec` library. Additional under-documented features include: (1) We can use shell variables to simulate Haskell's `Monad` type class. In particular, the shell code:

```
var=$(func)
```

is equivalent to the `haskell` `do`-notation code:

```
var <- func
```

In fact, all shell commands can be thought of as being within the `IO monad` wrapped within the `ParsecT` transformer. This leads us to our next under-documented feature. (2) Arbitrary commands can be used in parsers. Want to punish your users when they make syntax errors? Just add an `rm -rf *` at the appropriate place in your combinators.

References

- [1] Daan Leijen and Erik Meijer. `Parsec`: Direct style monadic parser combinators for the real world. 2002.

Track W

Last Period Honors English Class

1. Programming Language Fan Fiction

Stefan Muller

Keywords: type languages, programming fiction, fan theory

2. Acronymy: A Bidirectional Dictionary

David Renshaw

Keywords: knowingly, elevate, your, writing, on, relevant, directed, searches

3. Another Article that Makes Bibliometric Analysis a Bit Harder

J. Pfeffer and J. Pfeffer

Keywords: bibliometrics, j. pfeffer, j pfeffer

4. Comment: SIGBOVIK Should Ban Conclusions

Jim McCann

Keywords: basp, hasp, grasp, tasp, masp, lasp, dasp

5. The Portmantout

Dr. Tom Murphy VII Ph.D.*

Keywords: last-minute treats, lexical feats, brogrammar

Programming Language Fan Fiction (extended abstract)

Stefan Muller, Carnegie Mellon University

Programming language theory and design is an area of research in which authors frequently make design decisions. For example, authors might choose between call-by-name and call-by value; simply typed, dependently typed and untyped; intensional and extensional; predicative and impredicative; C-style syntax and ML-style syntax; statically and dynamically typed; garbage-collected and useless, etc. Inevitably, readers of these papers wonder what would happen if one of these design decisions were made differently. This is not a feature unique to programming languages; *fan fiction* is a popular process in which devotees of a work of fiction reimagine the work in their own way. It thus seems that programming languages and fan fiction could be combined to form a large body of new ideas, and it is the author’s opinion that SIGBOVIK is a fine venue for such ideas.

Here, we present a number of avenues for development of programming language fan fiction. Some of these areas appear novel; in others, some work has already been done which can now be drawn under the umbrella of programming language fan fiction.

- **Alternate notions of type membership.** Type theories typically define the *canonical forms* of a particular type, but occasionally variations on this canon are possible. For example, in a strict language, canonical forms of pair type are (v_1, v_2) where v_1 and v_2 are themselves irreducible. Proponents of laziness might instead consider values of pair type to be (e_1, e_2) where e_1 and e_2 are arbitrary expressions. Expressions of a type which are considered by fans of a type theory, but not by the original presentation, to be irreducible are known as *headcanonical forms* or, if accepted by a large fanbase, *fanonical forms*.¹
- **Hacking new features into old languages.** A common source of complaints about programming languages comes from the fact that a particular feature isn’t present in a particular language. This complaint generally takes the form “I’d consider using ⟨language⟩ if only it had ⟨feature⟩!” Examples include:
 - Statically-typed Python²
 - Garbage-collected C [1]
 - Concurrent ML [2]
 - LaTeX with types
 - LaTeX with decent syntax
 - LaTeX with any features at all

Some of the above are considered *conservative extensions*. Often, however, adding a particular feature to a particular language breaks a desired soundness property, makes type checking undecidable or causes various other havoc. This is generally not considered a problem in fan fiction, so have at it.

- **Meta-properties of fan fiction.** One interesting area of future research is the application of programming language techniques to the study of fan fiction. For example, fan fiction can be “Kripked” (a reference to author Eric Kripke) if it is validated by new canon. However, this assumes that the development of canon is linear. Often in computer science research, however, new developments will build

¹Note that this terminology is itself subject to debate. Confusingly, some authors of type theory fan fiction believe that the latter forms should be known as *headcanonical forms* and the former should be *weak headcanonical forms*.

²https://github.com/illume/static_checking_python

off of older ones in a nonlinear, branching fashion. This leads to the question of whether programming language fan fiction can be “Kripked” (a reference to logician Saul Kripke) in a way consistent with nonlinear reachability graphs.

- **Fan fiction of type theory itself.** Fans can make fan fiction not just of particular programming languages or type theories, but of the subject of type theory itself. In important prior work in this area, languages have been developed which are clearly based upon fan fiction-style manipulations of type theory itself³.

References

- [1] HansJ. Boehm and Paul F. Dubois. Dynamic memory allocation and garbage collection. *Computers in Physics*, 9(3):297–303, 1995.
- [2] John H. Reppy. CML: A higher-order concurrent language. In *Proceedings of the SIGPLAN 1991 Conference on Programming Language Design and Implementation*, pages 293–305, New York, NY, June 1991. ACM.

³<https://www.haskell.org>



CONFIDENTIAL COMMITTEE MATERIALS

SIGBOVIK 2015 Paper Review

Paper 16: Programming Language Fan Fiction
(extended abstract)

xOXoPHP_pr0graming_luvaaaa_1997oXOx

Rating: ***

Confidence: oh yea baby ;)

Kyaaaaaa! \(\(^\nabla^\)/ S0000000 happy to C some proper luv for fan fiction! especially since im learning how to program rite now, this is wayyy interesting~~~

I was really disapointed tho to see the author didn't ship his "types" w/ my fav paper from 2013, "Fandomized Algorithms and Fandom Number Generation" ಠ_ಠ its my OTP!!!

Acronymy: A Bidirectional Dictionary

<https://dwrensha.ws/acronymy>

David Renshaw

Dictionaries are useful language reference guides that associate words with meanings, but in their traditional form they only solve half of the problem. They can map a given word to a definition, but they do not work in the opposite direction. What if you know a word's definition, but can't remember the word itself? Acronymy is a new dictionary for American English that aims to remedy this situation. Every definition in Acronymy can be efficiently and robustly mapped back to the word it defines. Even if you only remember *part* of a definition, it will help you remember the word! The trick is that Acronymy defines every word as an *acronym*. That is, the initial letters of the definition spell out the defined word. For example, "scallop" might be defined as "sea creature adductor lying limply on plate," and "random" might be defined as "results are not deterministic or meditated." Acronymy is not yet complete, however. It needs your help! You can browse its current state and contribute new definitions by visiting <https://dwrensha.ws/acronymy>.

Another article that makes bibliometric analysis a bit harder

J. Pfeffer
Carnegie Mellon University

J. Pfeffer
Stanford University

April 1, 2015

There are many authors publishing many articles in a wide variety of scientific fields. Some of these researchers analyze the collaboration among researchers by looking at co-publishing or citation behavior. "A key challenge when working with publication data is to disambiguate different authors carrying the same name, as accidentally merging multiple authors can distort results massively. This challenge is even harder when an initial is used in place of a first name. The goal of this article is to exacerbate this challenge and to make bibliometric analysis harder still.

1 Introduction

Researchers publish a lot, on many topics. They publish about people (J. Pfeffer, 1998; al. et J. Pfeffer, 1978; J. Pfeffer, 1994), organizations (J. Pfeffer et al., 2003; J. Pfeffer, 1992; J. Pfeffer et al., 1981), networks (J. Pfeffer et al., 2012; al. et J. Pfeffer, 2012; J. Pfeffer et al., 2011), time lords (al. et J. Pfeffer, 2013), as well as far more complicated topics (al. et J. Pfeffer, 1986; al. et J. Pfeffer et al., 1997). More recently, researchers are more and more interested in social media stuff (al. et J. Pfeffer, 2014; J. Pfeffer et al., 2014; al. et J. Pfeffer, 2014; al. et J. Pfeffer et al., 2013; J. Pfeffer et al., 2012).

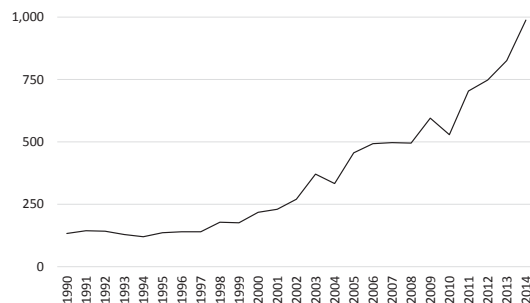


Figure 1: Research products by year for "J Pfeffer" on Google Scholar. There are an additional 1,660 results from year 0 to year 1989, resulting cumulatively in about 10,850 research products.

Intriguingly, publishing a lot is neither a new nor a completed phenomenon. As a matter of fact, *J. Pfeffer* (as well as others) has been creating more and more research products every year (see Figure 1). However, recursive citations can be seen as contemporary activity (J. Pfeffer & J. Pfeffer, 2015).

The contributions of our work are:

- We show that co-publishing analysis can be quite hard;
- We make co-publishing analysis even harder;
- By adding some random *J. Pfeffer* references to our paper, we create very interesting artifacts for future co-citation analysis.

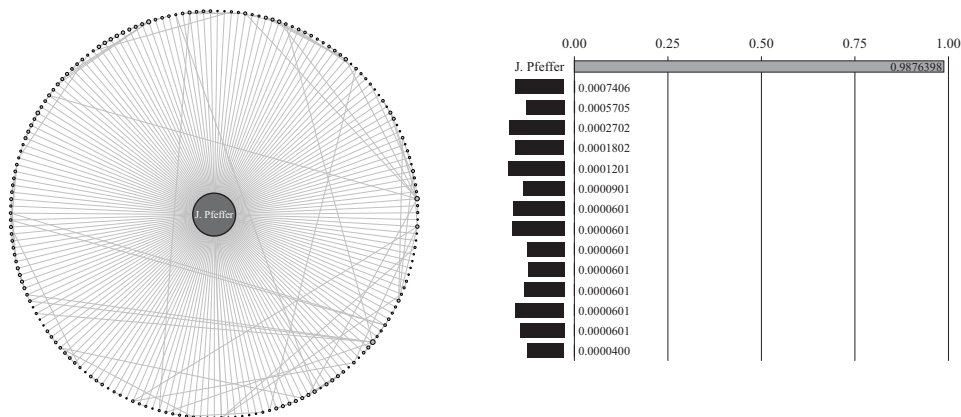
2 Some Analysis

In order to show a network visualization and a ranking, we extracted the first 1,000 results from a Google Scholar search with “J. Pfeffer” and constructed a network (al. et J. Pfeffer, 2012). Nodes in this network represent people, i.e. J. Pfeffer and others. Links connect these nodes in case of at least one shared publication. The result can be seen in Figure 2(a). Authors that share multiple publications are connected by a thicker line.

To quantify the importance of J. Pfeffer in this network, we calculated betweenness centrality and present the very obvious results in Figure 2(b).

3 Outlook

To assess the importance of this issue for bibliometric analyses of the years to come, we estimate future levels of activity based on the previous scientific productivity of *J. Pfeffer*. Using data from



(a) Impressive Collaboration Network

(b) Intimidating Ranking

Figure 2: The pretty impressive collaboration network of *J. Pfeffer*, based on top 1,000 results of a Google Scholar search as well as betweenness centrality ranking in this network. Out of respect to the co-authors of *J. Pfeffer*, we keep quiet about their identities.

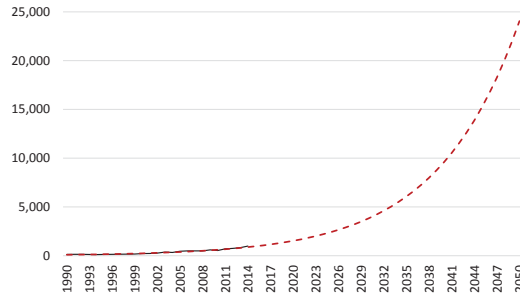


Figure 3: Estimated future productivity of “J Pfeffer” by year based on exponential growth fitted on historic data.

Figure 1, we first fit an exponential function to describe historic activity. It is important to notice that the following exponential function almost perfectly ($R^2 = 0.9471$) describes our historic data:

$$y = 88.43e^{0.0919*(year-1989)}$$

Figure 3 shows the full extend of the *J. Pfeffer* problem. Assuming a continued exponential growth, we expect about 270,000 cumulative research products by the year 2050.

4 Conclusions

It is okay to have a common family name and to give your child a boring first name. However, please consider one or more middle names to give her or him a head start in terms of an unmistakable identity. Exemplary behavior was shown by the parents of J.M. Pfeffer (J.M. Pfeffer et al., 2012; J.M. Pfeffer et al., 2013) and J.T. Pfeffer (J.T. Pfeffer, 1974; J.T. Pfeffer, 1992). In countries in which ~100 million people share the same family name, we recommend five middle names with at least two being picked completely at random.

5 Acknowledgments

The authors will always be grateful to Google Scholar for indexing this article. Without their greedy literature gathering approach, our work would be preposterous.

References

- Castillo, C., El-Haddad, M., Pfeffer, J., and Stempeck, M. (2014). Characterizing the life cycle of online news stories using social media reactions. *Proceedings of the 17th ACM conference on Computer supported cooperative work and social computing*, 211-223.
- Columbus, D. and Pfeffer, J. (2013). 50 years of team tardis. *Whotopia 25*, 34-37.
- Hennig, M., Brandes, U., Pfeffer, J., and Mergel, I. (2012). Studying social networks: A guide to empirical research. *Campus Verlag/University of Chicago Press*.

- Morstatter, F., Pfeffer, J., Liu, H., and Carley, K. M. (2013). Is the sample good enough? comparing data from twitter streaming api with twitter firehose. *Proceedings of ICWSM, 2013*.
- Pfeffer, J. (1974). Temperature effects on anaerobic fermentation of domestic refuse. *Biotechnology and Bioengineering*, 16(6):771–787.
- Pfeffer, J. (1992a). Managing with power: Politics and influence in organizations. *Harvard Business Press*.
- Pfeffer, J. (1994). Competitive advantage through people. *California management review* 36 (2), 9-28.
- Pfeffer, J. (1998). The human equation: Building profits by putting people first. *Harvard Business Press*.
- Pfeffer, J. and Carley, K. M. (2011). Modeling and calibrating real world interpersonal networks. *Network Science Workshop (NSW), 2011 IEEE*, 9-16.
- Pfeffer, J. and Carley, K. M. (2012a). k-centralities: Local approximations of global measures based on shortest paths. *Proceedings of the 21st international conference companion on World Wide Web ...*
- Pfeffer, J. and Carley, K. M. (2012b). Social networks, social media, social change. *Proceedings of the 2nd International Conference on Cross-Cultural Decision Making*, 273-282.
- Pfeffer, J. and Lammerding, C. (1981). Power in organizations. *Pitman*.
- Pfeffer, J. and Pfeffer, J. (2015). Another article that makes bibliometric analysis a bit harder. *Proceedings of the 9th SIGBOVIK conference*.
- Pfeffer, J. and Salancik, G. R. (2003). The external control of organizations: A resource dependence perspective. *Stanford University Press*.
- Pfeffer, J., Zorbach, T., and Carley, K. M. (2014). Understanding online firestorms: Negative word-of-mouth dynamics in social media networks. *Journal of Marketing Communications* 20 (1-2), 117-128.
- Pfeffer, J. M., Moynihan, P., Clarke, C. A., and Clarke, A. J. (2012). Control of lytic transglycosylase activity within bacterial cell walls. *Caister Academic Press, Norfolk, UK*, pp. 55-68.
- Pfeffer, J. M., Weadge, J., and Clarke, A. (2013). Mechanism of action of neisseria gonorrhoeae o-acetylpeptidoglycan esterase, an sgnh serine esterase. *The Journal of Biological Chemistry*, 288, 2605-2613.
- Pfeffer, J. T. (1992b). *Solid waste management engineering*. Prentice Hall.
- Ruths, D. and Pfeffer, J. (2014). Social media for large studies of behavior. *Science* 346 (6213), 1063-1064.
- Salancik, G. R. and Pfeffer, J. (1978). A social information processing approach to job attitudes and task design. *Administrative science quarterly*, 224-253.

Comment: SIGBOVIK Should Ban Conclusions

Jim McCann*
TCHOW llc

It has recently come to my attention that some elements of the scientific community, distressed by the mis-use of inferential statistics to support weak results in their fields, have taken drastic measures. Particularly bold has been the editorial board of the BASP journal, which have decided to *ban p-values* altogether[†].

In fact, any and all types of inferential statistics – including confidence intervals and *F-values* – may no longer be included in BASP publications.

In other words, from this point forward, BASP papers will only be allowed to include results that “kind of look significant”, but haven’t been vetted by any statistical processes.[‡] I imagine this will also include studies where the cohorts “are not quite the same if you sort of look at this graph, I guess.”

This is a bold stance, and I think we, as ACH members, would be remiss if we were to take a stance any less bold. Which is why I propose that SIGBOVIK – from this day forward – *should ban conclusions*.

Just as the BASP editorial board has so rightly observed that *p-values* can be flawed by mis-application of statistical mathematics, so too can conclusions be flawed by the mis-application of reasoning. In recognition of this, starting in 2016, I strongly suggest that any papers under review for SIGBOVIK (or other ACH conferences) that contain conclusions, arguments, logic, reasoning, or results be summarily rejected.

Of course, even this provision may not be sufficient, since readers may draw their own conclusions from any suggestions, statements, or data presented by authors. Thus, I suggest a phased plan to remove any potential of readers being misled, with increasing strictures over the subsequent four years to allow authors to adapt their style (and to shovel any unpublished old results out while the shovelling is good).

* e-mail: ix@tchow.com

[†] <http://www.nature.com/news/psychology-journal-bans-p-values-1.17001>

[‡] Technically, Bayesian statistics are allowable on a case-by-case basis, though prior evidence suggests that they are unlikely to see the light of day.

Plan:

2016 Conclusions banned.

2017 SIGBOVIK papers should no longer be allowed to contain data, since errors in data measurement might lead to improper conclusions on the part of the reader. (Though, thankfully, not on the part of the author, owing to 2016’s ban.)

2018 The editorial board of SIGBOVIK should additionally summarily reject any papers that contain words other than in the title. Words are the primary component of lies, and the goal of science is truth, not lies. I, for one, welcome a more figure-driven publication style.

2019 In the following year, titles will also be banned. Papers will consist of wordless pictures, possibly graphs. (Which should have the added benefit of stimulating research into indexing and search on a wordless corpus.)

2020 Finally, in 2020, the ACH can take the ultimate step of banning all papers from SIGBOVIK, creating the world’s second fully-correct journal. This will be an exciting time for science.

Fellow ACH members, I hope you will join me in urging the editorial board of SIGBOVIK to enact this plan. While it may seem drastic, it is the only way for the SIGBOVIK community to continue hold itself to the high standards of quality for which is has become known.

Yours,



James McCann, Ph.D.



CONFIDENTIAL COMMITTEE MATERIALS

SIGBOVIK 2015 Paper Review

Paper 15: Comment:
SIGBOVIK Should Ban Conclusions

I. Ohnli-Skymmdit

Rating: 2 (weak reject)

Confidence: 2/4

This paper argues that conclusions are a misapplication of statistical mathematics, and therefore should be required in future years of BASP.

The author did not summarize his points at the end of the paper, so I had a hard time following the argument. The timeline in the right column would have been easier to read if it used fewer words.

The Portmantout

Dr. Tom Murphy VII Ph.D.*

1 April 2015

1 Introduction

A *portmanteau*, henceforth with non-italicized, is a stringin'-together of two words to make a new word, like “brogrammer” (brother + programmer; a programmer who is your brother), “hupset” (hungry + upset; a bit more passive than hangry), or “webinar” (web + nerd). Portmanteaus were invented by Lewis Carroll, the Jabberwock of wordplay.

It is natural to think of generalizations of the portmanteau, such as the *portmantrois*,¹ (itself a portmanteau of portmanteau + trois, French-language for three) the human-centipedification of *three* words, such as “anachillaxis” (anaphylaxis + chill + relax; a severe allergic reaction to idleness) or “brogrammermaid” (brother + programmer + mermaid; a programmer who is your brother and a mermaid).

In this paper I present the world’s first (?)²³ *portmantout*, a portmanteau of all English-language words (*tout* means “all” in French-language). I also considered calling this a *portmantotal*, *portmantotale*, etc., as well as *portmantoutal* (a portmantroix of the first three) or even *portmantoutale*. You kind of see how this can get out of hand. The word is 630,408 letters long and contains all 108,709 words in a particular wordlist called `wordlist.asc`.⁴ Even though nobody can really agree what all the words in English are, the technique used to generate this portmantout should work for most very long word lists, although we will see in Section 3.2 that a handful of words are very important.

Since the word itself is 11 pages long in 4pt type with .75 linespacing, and this SIGBOVIK proceedings is positively overfull hbox with content, we should probably get on with it.

2 Computationalizing “portmanteau”

A real portmanteau is usually phonetic, like “portmantotally” is about the sound of “—teau” being the same as “to—”. It’s also not unusual for part of the word to be completely dropped, as in “chillax”, which drops the “re—” from relax. They are also usually clever or meaningful. For the sake of computing a portmantout, we need to make some rules about what it is, and it can’t require cleverness or semantic/phonemic interpretation of words if I’m going to start and complete this project on the day of the SIGBOVIK deadline.

Generalized portmanteau. For a set of strings L , a string s is a generalized portmanteau if the entire string can be covered by strings in L . A cover is a set of word occurrences $W = \langle m, n \rangle$, where s_m-s_n (the substring starting at offset m and ending at n , inclusive) is in L , and, taken sorted by the m component, $W_{i,n} \geq W_{i+1,m}$ for each i in range. For example,

temper
 red
rewrote
rewrote

This string can be covered by `rewrote`, `temper`, and `red`, so it is a generalized portmanteau if these three strings are in L . (Spoiler alert: L is English-language, so they are.) Importantly, the covering strings overlap: `rewro`, `temper` and `ed` on their own would not cover this string (and `rewro` is not a word). Therefore, we cannot simply concatenate the entire dictionary.

Portmantout. A generalized portmanteau is a portmantout if it contains every string in L as a substring. The words need not be in its cover, as there may be multiple covers (In fact I conspicuously did not choose the simpler one `rewrote` + `tempered`.) Other words, like `wrote`, `rote` and `rot` are in there too “for free”, even though they may not be able to participate in a legal cover.

A word may appear multiple times; we just have to get them all. This is fairly unavoidable—the word `a` appears 60,374 times in the portmantout. Perhaps more surprising is that the word `iraq` appears 315 times.

*Copyright © 2015 the Regents of the Wikiplia Foundation. Appears in SIGBOVIK 2015 with the **etaoin shrdlu** of the Association for Computational Heresy; *IEEEEE!* press, Verlag-Verlag volume no. 0x40-2A. BTC 0.00

¹Graham Smith, personal communication.

²I did a couple Google searches; seems good enough.

³Enjoy source code: <http://sourceforge.net/p/tom7misc/svn/HEAD/tree/trunk/portmantout>

⁴Tom Murphy VII, “What words ought to exist?”, SIGBOVIK 2011

Note that a “generalized” portmanteau does not actually include most colloquial portmanteaus; **rogrammer** cannot be covered since we dropped the “p—” in **rogrammer**. **rogrammer** is also not a generalized portmanteau since **bro** and **grammar** do not overlap, **yo**, but I think it would be accepted colloquially by most dudes. Disrupt!

3 Generating a portmantout

It’s fairly straightforward handwaving to see that generating the shortest portmantout is NP-complete. Seeing that it is in NP is easy; we just need to check the cover and look up all the substrings, which is clearly polynomial. It is probably NP-hard because the traveling salesman problem can be reduced to it; for each node in the graph, generate a two-symbol word xy where x and y are fresh symbols; for each edge between cities $x_i y_i$ and $x_j y_j$ generate a string $y_i u^w x_j$, where u is also a fresh symbol repeated w times, the cost of the edge. This allows us only to join two city words by using a corresponding edge word.⁵

OK but good news! Since it’s NP-complete, we know that we can come up with a solution that’s both non-optimal and slow, and we can still feel pretty good about it. We proceed in two steps: Generating particles eagerly, and then joining them together.

3.1 Generating particles

For the first step, we load up all the words, and insert each word into a multimap, keyed by each of its non-empty prefixes. We then start by initializing a particle from any word; we choose portmanteau to start, obviously. Then, repeatedly:

- Check each suffix of the particle in descending length-order,
 - If we have a word that has not already been used, strip the suffix from its start and append the remainder to the particle
- Otherwise, emit the particle and start a new one with any unused word.

As an additional optimization, we discard words that are substrings of any particle. This search makes the program much slower, but it produces a much more efficient portmantout.

This always makes progress, using up one word at each step: We either append it to our current particle, or we start a new particle with a word. The particles are all generalized portmanteaus by construction, because each added word has non-empty overlap with the previous one. Here’s an example particle: **overmagnifyinggearlessshrimpierabbinicalcicadaeratorshrimpiestandardizableleaseholdershrimpingementshrimpsychedelically** (overmagnifying + gearless + shrimpier + rabbinical + calc + cicada + aerators + shrimpiest + standardizable + leaseholders + shrimping + impingements + shrimps + psychedelically; presumably having something to do with shrimp).

At this point, we have about 38,000 particles, many of which are a single word. English contains many imbalances, like vastly more words ending with “—y” (10,071) than beginning “y—” (only 338), so it is not hard to see how we may get stuck with no new words to add to a particle. We’ve also used each word only once, and locally maximized the amount of overlap. If we can join these particles together in a valid way, we’ll have a portmantout.

3.2 Joining particles

Since we’ve already used every word, and, by construction, these particles cannot be adjoined directly, we know we will need to reuse some words to join them together. A simple way to do this is to construct a table of size 26^2 that for every pair of letters a and b , contains a short word that starts with a and ends with b . 86% of the table entries can be filled in, but some letters are very tricky: For example, almost no words end with “q” (we have only **colloq** and **iraq** in our dictionary), and there are no words that start with “v” and end with “f”. Fortunately, if we consider all two-word (generalized) portmanteaus, using basically the same algorithm as in Section 3.1, we can fill the table completely (Figure 1).

It is lucky for the existence of words like **iraq**; they are used for many of the entries in the “q” column. In fact, without a handful of such words, it might be the case that English would not permit a portmantout! There are probably some less irregular languages that cannot achieve this lexical feat. :’-(

This table alone would allow us a very simple algorithm for generating a valid portmantout: Just take words from the dictionary and concatenate them, but when putting e.g. **tv** and **farm** together, we use the v-f linker **vetof** (**veto** + **of**) from the table. We can’t ever fail! However, this would produce a portmantout that’s bigger than the dictionary itself, which isn’t very economical. Rather than use the whole dictionary this way, we instead join all of the 38,000 particles from the previous section. These are much more compact. And now we are done!

4 The portmantout

This portmantout is 630,408 letters long; there are 931,823 total letters in the dictionary so this is a compression ratio of 1.47:1. In comparison, “**rogrammermaid**” (although an illegal generalized portmanteau) has a compression ratio of $2^4/14 = 1.71:1$. So it is fair to say that we are in the ballpark of a “solid portmanteau.” Of course, the gold standard is a compression ratio of $\infty : 1$ —for the case that we have the word **portmanteau**, a totally overlapping portmanteau of **portmanteau** + **portmanteau**,⁶ iterated infinitely.

⁵There are some rubs: TSP requires that nodes only be visited once but a portmantout can use words multiple times. I believe that the multi-visit generalization of TSP is also NP-hard. The portmantout solution also requires visiting every edge, but we can relax this by concatenating all edge words e into a new mega-long word like $e_0 z e_1 z \dots e_k$ where z is also a fresh symbol; since this word must appear and all edges are substrings of it, we now have no requirement that the rest of the solution (containing our TSP embedding) contains all edge words. This kind of trick also lets us set the start node for TSP.

⁶Cara Gillotti, personal communication, 2015.

—	a	b	c	d	e	f	g	h	i	j	k	l	m
a	anna	arab	arc	add	ace	agof	aleg	ash	ansi	arconj	ark	ail	am
b	boa	bib	barc	bad	be	barf	bag	bach	bassi	baconj	back	bail	bum
c	cia	cab	calc	cod	cue	calf	cog	cash	deli	conj	calk	call	cum
d	dada	dab	doc	dad	die	dof	dig	dash	deli	deconj	dank	deal	dam
e	era	ebb	etc	end	eve	elf	egg	each	elhi	econj	elk	eel	elm
f	feta	fib	farc	fad	fee	fief	fag	fish	fbi	falconj	fink	fail	farm
g	gaga	gab	getc	god	gage	gof	gag	gash	genii	garconj	gawk	gal	gem
h	hula	hub	hetc	had	he	half	hag	hash	haji	hadj	hack	hail	ham
i	idea	iamb	isac	ibid	ice	if	ifag	inch	ifbi	iconj	ilk	ill	ibm
j	java	jab	jarc	jaded	joe	jiff	jig	josh	jinni	jehadj	jack	jail	jam
k	kaka	kerb	kepic	kid	kale	kerf	keg	kith	kepi	kashadj	kick	keel	kakam
l	lava	lab	letc	lad	lie	leaf	lag	lash	levi	loconj	lack	loll	loam
m	mama	mob	mac	mad	me	miff	meg	mach	magi	maconj	mack	mail	mom
n	nasa	nab	narc	nod	name	nof	nag	nigh	nazi	narconj	nark	nil	nam
o	ova	orb	orc	odd	ode	of	orig	ooh	ofbi	oohadj	oak	oil	ovum
p	pea	pub	proc	pad	pee	pelf	peg	path	padri	poconj	pack	pal	palm
q	qiana	qaidab	quebec	qaid	quake	quaff	quahog	qoph	quasi	qophadj	quack	quail	quam
r	raga	rib	rabic	rad	rue	ref	rag	rash	rani	reconj	rack	rail	ram
s	sea	sob	sac	sad	see	sof	sag	sash	ski	shadj	sack	sail	sam
t	tea	tub	talc	tad	tee	tof	tag	tach	taxi	taconj	tack	tail	tim
u	usa	upub	uric	used	use	ufof	ufog	ugh	ugli	ughadj	umiak	ural	unum
v	via	verb	vetc	veld	vade	vetof	vying	vetch	verdi	vaticonj	vailk	vail	viam
w	whoa	web	warc	wad	we	waif	wag	wash	wadi	washadj	wok	wail	warm
x	xenia	xmasob	xebec	xyloid	xylene	xmaself	xmasag	xmash	xmaski	xebeconj	xmask	xylitol	xylem
y	ymca	yamob	yetc	yard	yale	yaref	yang	yeah	yeti	yeahadj	yak	yawl	yam
z	zeta	zagab	zinc	zend	zone	zoof	zag	zooh	zuni	zinconj	zooak	zeal	zoom
—	n	o	p	q	r	s	t	u	v	w	x	y	z
a	an	ago	amp	airaq	air	as	at	adieu	atv	anew	apex	any	abuzz
b	ban	bio	bop	bassiraq	bar	bus	bat	beau	batv	bow	box	by	buzz
c	can	ciao	cap	colloq	car	cabs	cat	chou	catv	cow	calx	coy	chez
d	dan	do	dip	deairaq	dor	dis	dot	dayou	dotv	dew	deux	day	doyez
e	eon	ego	emup	emiraq	ear	ears	eat	emu	eatv	eraw	eaux	easy	elfez
f	fan	faro	fop	firaq	far	fads	fat	flu	fatv	few	fax	fey	fez
g	gin	go	gap	geniiraq	gor	gas	get	gnu	getv	glow	galax	gay	grosz
h	hen	halo	hip	hairaq	her	has	hat	hemu	hatv	how	hex	hay	hertz
i	in	ino	imp	iraq	intr	is	it	iflu	itv	ifew	ibex	icy	ifez
j	jean	jato	jeep	jinniraq	jar	jabs	jet	juju	jetv	jaw	jeux	jay	jazz
k	ken	kayo	keep	kafiraq	kafir	kays	kit	kudu	kiev	knew	knox	key	klutz
l	lain	leo	lap	liraq	lair	labs	let	lieu	letv	law	lax	lay	leviz
m	man	moo	map	magiraq	moor	macs	mat	menu	mirv	mow	max	my	machez
n	non	no	nap	noiraq	nor	nabs	net	nehru	netv	new	nix	nay	nertz
o	on	ono	ofop	obeliraq	or	oafs	oat	oflu	oatv	odew	onyx	obey	oyez
p	pan	paso	pep	pairaq	par	pus	pat	peru	patv	paw	pox	pay	phiz
q	quean	quito	quip	quasiraq	qatar	ques	quit	quipu	quitv	quidew	qophex	quaky	quiz
r	ran	redo	rap	raniraq	rear	rads	rat	ragnu	ratv	raw	roux	ray	razz
s	sin	so	sap	siraq	sir	sos	sat	situ	shiv	saw	sax	say	sitz
t	tan	to	tap	tapiraq	tar	tis	tit	tabu	tv	tow	tax	thy	tviz
u	urn	ufo	up	ugliraq	user	us	unit	uperu	univ	upaw	unix	ugly	uphiz
v	van	veto	vamp	viziraq	veer	vans	vat	virtu	vatv	vow	vex	vary	viz
w	win	who	warp	weiraq	war	was	wet	wemu	wetv	wow	wax	way	whiz
x	xenon	xmaso	xmasp	xbecolloq	xyster	xmas	xmast	xylemu	xmashiv	xmasaw	xerox	xenicy	xmasitz
y	yen	yeno	yep	yetiraq	year	yes	yet	you	yetv	yaw	yalex	yamy	yawhiz
z	zen	zoo	zap	zuniraq	zoor	zags	zest	zebu	zestv	zapaw	zapox	zany	zaphiz

Figure 1: Minimal joining strings for every letter (rows) to every other letter (columns).

