

THE ASSOCIATION FOR COMPUTATIONAL HERESY

PRESENTS

A RECORD OF THE PROCEEDINGS OF

SIGBOVIK 2013

*The seventh annual intercalary robot dance party in celebration
of workshop on symposium about Harry Q. Bovik's 2⁶th birthday*

Carnegie Mellon University
April 1, 2013



Association for Computational Heresy



Advancing computing as Tomfoolery & Distraction

SIGBOVIK

A Record of the Proceedings of SIGBOVIK 2013

ISSN 2155-0166

April 1, 2013

Copyright is maintained by the individual authors, though obviously this all gets posted to the internet and stuff, because it's 2013.

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, see <http://sigbovik.org/> for details.



SIGBOVIK 2013

Message from the Organizing Committee

It is with great pleasure, honor, enjoyment, cheer, verve, rapture, amazement, shock, and fleeting disgust that we present to you the proceedings of the Seventh Annual Intercalary Workshop about Symposium on Robot Dance Party of Conference in Celebration of Harry Q. Bovik's (2⁶)th Birthday. This has been a record breaking year in innumerable many ways:

1. Seven is, to our knowledge, the highest annuality of this workshop.
2. The record for most submission with effectively the same name was broken this year.
3. We had our fewest organizing committee coups. Zero!
4. We have the longest SIGBOVIK paper ever, weighing in at a mighty twenty-two pages.
5. We tied our record for most records tied or broken in a single year. (This record is only speculative.)

However this year, for all its ups, was not without its downs. We got off to a rocky start when the entire organizing committee neglected all work in expectation of the apocalypse on December 21, 2012. Once that turned out not to be an issue, the entire organizing committee partied through January and was hungover through February on homeward flight. But once March rolled around, we were good to go! And go we did.

How to use these proceedings

Sure you could read them (see next section), but have you ever considered that these proceedings could also work to: fix a wobbly table, wrap a fish, hold down papers, invalidate a hypothesis, fan yourself, fan others, wipe grime from your forehead, wipe grime from your bottom, wipe that grin off your face, polish steel, flavor coffee, steal polish, self-flagellate, break a window, underestimate the value of a really fine set of encyclopedias, scrub a duck, build a fragile parachute, reflect your inner turmoil, provide kindling for a fire, shade a lizard, be a great success (though perhaps overburdened with symbols in – I would say – the surrealist tradition), provide catharsis, position your keyboard or monitor ergonomically, fill a bookshelf, validate your self-image, expose conspiracies, break the bank, or provide eternal life?

How to read these proceedings

SIGBOVIK is a wide-ranging conference, at least as moreso this year than other years; therefore, for your browsing convenience, the papers this year have been divided into six tracks.

Tracks may be read separately, together, or – special this year – in time-trial mode, in which you will not be reading against other scientists but (rather) against the clock. Read each track for three laps, and try to beat your best time, or compete against your friends using our social integration feature. Simply sign in below to enable leaderboards, tap-to-tweet, live bookmarks, pop-up notifications, and more:



By signing in with your facebook or twitter account you empower the ACH and its designated subsidiaries, lackeys, thugs, and/or flunkies to make important life decisions on your behalf, including – but not limited to – hair color, style of dress, accent, and what you had for lunch yesterday. The content of these decisions will be reported in tweets or postings reputed to come from you. ACH may, at its sole discretion, decide to notify you in advance of these postings. In any event, you will will be required to conform to the content of these declarations within twenty-four hours of their appearance or risk confusing your friends. If you choose not to comply, a lifestyle/fashion strike team may be deployed. ACH is not liable for anything, ever. Not our chairs, not our problem. ¹

Massage from the organizing committee

Gosh, you look tense. Let us sooth those muscles for you. Simply rub this proceedings on your back in slow circles. Doesn't that feel better?

¹Thanks <http://www.komodomeia.com/blog/2009/06/sign-in-with-twitter-facebook-and-openid/>



SIGBOVIK 2013 Paper Review

Paper 0: Message from the Organizing Committee

L. Johnson

Rating: 4 (strong accept)

Confidence: 1/4

This is a good paper. But I don't get the bit about homeward flight.

TABLE OF CONTENTS

Track 1: Contemporary Issues in Politics and Law

1. SIGBOVIK License Agreement (excerpt)	2
2. Redistributive Version Control Systems.....	3
3. Psychology in Response to Presidential Poles.....	8

Track 2: Harry Bovik's Research Calculus

1. The Randomly-Scoped Lambda Calculus.....	22
2. Recovered Mathematical Journal	29
3. MacLeod Computing: A New Paradigm for Immortal Distribution	33
4. The $(\infty,1)$ -Accidentopos Model of Unintentional Type Theory (extended abstract)	37

Track 3: Artificial Stupidity

1. You Only Learn Once: A Stochastically weighted AGGRegation Approach to Online Regret Minimization	43
2. I Lost The Game, and So Will You: Implications of Mindvirus Circulation in a Post-Singularity World.....	47
3. Fandomized Algorithms and Fandom Number Generation	51

Track 4: Computer Vision(aries)

1. Optimal Coupling and Gaybies	54
2. Cat Basis Pursuit	59
3. A Spectral Approach to Ghost Detection.....	64

Track 5: Productivity and Meta-Productivity

1. Really Amazing New Idea.....	70
2. METHOD AND APPARATUS FOR PRESSING SPACEBAR	75
3. METHOD AND APPARATUS FOR PUSHING SPACEBAR.....	76
4. Find a Separating Hyperplane With This One Weird Kernel Trick (sponsored contribution).....	77
5. On n -Dimensional Polytope Schemes	78
6. DUI: A Fast Probabilistic Paper Evaluation Tool.....	82
7. Paper and Pencil: a Lightweight WYSIWYG Typesetting System	88

Track 6: Time Travel, Space Travel, and Other Fun Games for Children

1. A Proposal for Overhead-Free Dependency Management with Temporally Distributed Virtualization.....	92
2. The n -People k -Bikes Problem.....	96
3. The Problem of Heads of a Fighting Force from Long Ago.....	101
4. duoludo: a game whose purpose is games.....	111
5. The First Level of Super Mario Bros. is Easy with Lexicographic Orderings and Time Travel ... after that it gets a little tricky.	112

TRACK 1

Contemporary Issues in Politics and Law

1. SIGBOVIK License Agreement (excerpt)

Ix Tchou, Ph.D.

2. Redistributive Version Control Systems

Karlo Angiuli and Frederick Engels

Keywords: redistribution, communism, version control

3. Psychology in Response to Presidential Poles

Timothy Broman

Keywords: statistics, politics, election, cognitive dissonance, confirmation bias

An Excerpt from:
The SIGBOVIK License Agreement (Proposed)*

1 Definitions

(...)

Y. If this clause is said to apply to any other clause (hereafter, the *Subject*), it will be as though the *Subject* shall have no effect as originally written, and the following will pertain in its stead:

(a) Let the clause following this one be known as the *Target* clause, and replace it with the following subclauses:

i. This clause shall have the same force as the *Subject* clause.

A. This clause shall have the same force as the *Target* clause.

B. This clause shall have the same force as the *Target* clause.

(b) Let the clause following this one be known as the *Target* clause, and replace it with the following subclauses:

i. This clause shall have the same force as the *Subject* clause.

A. This clause shall have the same force as the *Target* clause.

B. This clause shall have the same force as the *Target* clause.

(...)

7 Agreement

1. All parties must read and understand the content of this agreement.

(...)

5. Let the term *Binding Year* refer to 2013, unless redefined in subsequent clauses. The following subclause is subject to provision 1.Y:

(a) This agreement shall be binding in the *Binding Year*. In addition, in any subclauses of this clause, let the term *Binding Year* refer to the year following the current *Binding Year*.

(...)

*As prepared by ix@tchow.com , PhD.

Redistributive version control systems

Karlo Angiuli Frederick Engels

March 18, 2013

Abstract

A spectre is haunting Github; the spectre of communism. Distributed version control has led to a new era of free software, but commit inequality remains rampant. We describe a system in which programmers code according to their abilities, on repositories according to their needs.

1 Open-source bourgeois and proletarians

The history of all hitherto existing software is the history of class struggles.¹

iTunes and Winamp, Intel C++ Compiler and Borland C++, Adobe Illustrator and Adobe FreeHand, Microsoft Word and Corel WordPerfect, in a word, **oppressor and oppressed**, stood in constant opposition to one another, carried on an uninterrupted, now hidden, now open fight, a fight that each time ended in the ruin of a contending software package.

In the closed-source epochs of history, we find almost everywhere a complicated arrangement of software into various orders, a manifold gradation of popularity. The modern open-source society that has sprouted from the ruins of closed-source society has not done away with class antagonisms. It has but established new forms of struggle in place of the old ones.

Our epoch, **the epoch of free software**, possesses, however, this distinct feature: it has simplified class antagonisms. The free software movement, during its rule of scarce thirty years, has created more massive and more colossal development forces than have all preceding generations together.

But the successful projects, the **Open-Source Bourgeoisie**, have called into existence the coders who are to bring their own demise—the **Free Software Proletariat**, the developers of unsuccessful free software.

The lower strata of free software—the GNU Hurd, GNU arch, and Guile—all these sink gradually into failure, partly because **their diminutive development activity does not suffice for the scale** on which Modern Software, like Linux, git, and Emacs Lisp, are developed.

¹The history of all hitherto existing object-oriented programming is the history of `class` struggles.

With their failure begins their struggle with the open-source bourgeoisie. At first the contest is carried on by individual developers, then by online communities, against the individual software projects which succeed them.

The proletariat direct their attacks not against the open-source conditions of development, but against the other projects themselves; they **argue on Internet forums**, they **complain on listservs**, they seek to restore by force the vanished status of the developer of Lesser-Known Software, the Software Proletariat.

At this stage, the developers still form an incoherent mass scattered over the whole Internet, broken up by their mutual competition. But with the development of social version control, the developers not only increase in number; they become concentrated in greater masses. Thereupon, the developers begin to form communities.

Altogether collisions between the classes of the old society further, in many ways, the course of development of the open-source proletariat. **The open-source bourgeoisie finds itself involved in a constant battle.** At first with the closed-source community; later on, with those portions of the open-source bourgeoisie itself, whose interests have become antagonistic to the progress of software.

In all these battles, it sees itself compelled to appeal to the free software community at large. The bourgeoisie itself, therefore, supplies the proletariat with its own elements of computer science and social education, in other words, it **furnishes the proletariat with knowledge for overcoming the bourgeoisie.** What the bourgeoisie therefore produces, above all, are its own grave-diggers. **Its fall and the victory of the proletariat are equally inevitable.**

2 Redistributive version control

The immediate aim of **redistributive version control systems** (RVCS) is formation of all open-source developers into a class, overthrow of the open-source bourgeois supremacy, **conquest of all software development** by the proletariat.

The Open-Source Revolution overthrew proprietary software in favor of free software. The distinguishing feature of RVCS is not the abolition of free software development generally, but the abolition of bourgeois free software projects. But modern bourgeois software is the final and most complete expression of the free software movement. In this sense, the theory of RVCS may be summed up in the single sentence: **Abolition of anomalously successful software.**

Specifically, the RVCS project aims:

1. To abolish all other version control.
2. To require that at least **30% of each developer's commits**, by diff line, must be to less fortunate repositories than one's own.
3. To displace the free software bourgeoisie, the Torvaldses and Shuttleworths of the world, from their dominion over successful software projects.

Under an RVCS software development paradigm, the community will choose as one those projects worthy of Communal development. This represents a significant streamlining of the bourgeois software development model, and will ensure an abundance of software to satisfy the needs of all developers. Thus shall developer person-hours be allotted **to each repository according to need, from each developer according to their ability.**

RVCS deployments will be federated under the dictatorship of the proletariat; at the end of each working day, each developer shall receive a certificate that they have furnished such-and-such a number of lines coded to socially-mandated repositories, and with this certificate, they may commit an appropriate number of lines to repositories of their own choice.

Somebody invested in libraries and runtimes. **If you've got a program, you didn't build that.** Somebody else's compiler made that happen. —Barack “Redistribution” Obama

3 Implementation issues

We acknowledge that, even in a RVCS utopia, the bourgeoisie may wish to steal for themselves the contribution they owe the developers of the world. Such **redistribution avoidance** will take many forms.

Such developers might:

1. establish their repositories in offshore jurisdictions (GitHub:Monaco, Bit-seau de Seychelles);
2. perpetually fork their own repositories, or set up shell repositories, in order to disguise their own commit history;
3. avoid newlines in their own repositories, artificially decreasing their perceived net worth; or
4. secret their repositories on hard drives, distributing releases via carrier pigeon.

Techniques for discouraging this selfishness will be explored in future work.

Acknowledgements

Apologies to Marx and Engels' *The Communist Manifesto*, 1888 translation by Samuel Moore.²



²<http://www.marxists.org/archive/marx/works/1848/communist-manifesto/index.htm>



SIGBOVIK 2013 Paper Review

Paper 18: Redistributive version control systems

Team Lead Benedict XVI Rating: 0 (strong reject)

Confidence: 4/4

The authors forgot that programmers always remain programmers. They forgot programmers and they forgot programmers' freedom. The authors forgot that freedom always remains also freedom for bad code. The authors thought that once the version control system had been put right, everything would automatically be put right. Their real error is materialism: programmers, in fact, are not merely the product of version control conditions, and it is not possible to redeem them purely from the outside by creating a favourable development environment.

Psychology in Response to Presidential Poles

The election campaign of 2012 was notable for many reasons. It featured an unprecedented third-party spending onslaught brought about by the easing of campaign finance restrictions. It featured more negative advertising and rhetoric than any other recent presidential campaign. And it featured a bevy of political poles and statistical meta-analyses aggregating those poles.

The continued growth of the Internet has made more and more polls available to others than those who produce it. As more polling organizations publish their results in an online, easily disseminable format, there has been a corresponding growth in the number of statisticians attempting to “average” these polls and predict even more accurately the outcome of the election. It turned out that the statistical methods used produced very accurate predictions of the results, particularly in the case of Nate Silver, a political blogger for the New York Times. His forecast correctly predicted the winners of all 50 states and the District of Columbia, and his 95% confidence intervals were correct on 48 out of 51 cases—that is, as close to 95% as it is possible to be with 51 trials. However, in the run-up to Election Day, the methods of Mr. Silver and his colleagues were subjected to severe scrutiny by conservative commentators, who believed that support for the Republican candidate, Mitt Romney, was several inches higher than what was being reported by the polls and analyses. Those commentators also made personal character attacks against some of those who predicted an Obama victory based on polling.

Polls are supposed to be an unbiased predictor of the outcomes of elections. Their success in this regard is dependent on the methodologies employed in collecting the polls. Some methodologies may produce polling biases, either generally or in particular circumstances. As a result, there are many potential reasons to discredit a particular poll or meta-analysis, and political commentators usually find them. However, the discreditations cast by political

commentators often show polemical bias as well as confirmation bias and cognitive dissonance. This paper seeks to examine some of these lapses in a psychological context as well as a historical one.

The first conflict arises in the polling itself. How does one sample the electorate? There is no easy way to choose a random sample among all **voting**-eligible U.S. citizens. In the U.S. presidential election system, it is more valuable to have the results of the state polls, anyway, thanks to the electoral college. But this doesn't defeat the sampling problem, it only breaks it into fifty smaller, but still immense, problems. There are no perfectly accurate voter lists, and while there are many reasonably accurate ones, the real question is: how do we contact these people? Calling landlines will result in a more conservative demographic compared to cellphones,¹ and Internet polling is relatively new and untested—what is its response rate and demographic?

And once you've tackled that, you have to get into the question of who you should really be talking to. Many polls sample “likely voters”, a subset that is intended to be representative of the voters who turn out on election day. However, Gallup, which has one of the most sophisticated likely-voter models,² has been consistently inaccurate in recent years and subject to high volatility of results³ that have been blamed on its likely voter model.⁴ The problem with a likely voter model is that voters may self-report as more or less likely to vote in the weeks before Election Day, entering or exiting the likely-voter pool and thus shifting the projected electorate without shifting their views.

To assess whether the sample used by a poll is representative of the electorate, some

-
- 1 Pew Research Center. “Cell Phones and Election Polls: An Update.” PewResearchCenter Publications, 13 October 2010. <http://pewresearch.org/pubs/1761/cell-phones-and-election-polls-2010-midterm-elections> retrieved 8 December 2012.
 - 2 Gallup. “Understanding Gallup's Likely Voter Procedures for Presidential Elections.” <http://www.gallup.com/poll/111268/how-gallups-likely-voter-models-work.aspx> retrieved 8 December 2012.
 - 3 Silver, Nate. “Gallup vs. the World.” *The New York Times*. <http://fivethirtyeight.blogs.nytimes.com/2012/10/18/gallup-vs-the-world/> retrieved 8 December 2012.
 - 4 Erikson, Robert S., Costas Panagopoulos and Christopher Wlezien. “Likely (and unlikely) voters and the assessment of campaign dynamics.” *Public Opinion Quarterly*, Vol. 68, No. 4, Winter 2004, 588-601.

commentators look at the proportions of self-reported party identification in the sample. In a standard version of the question, respondents are asked, “do you consider yourself a Republican, a Democrat, an independent, or what?”⁵ Unfortunately, party identification is a bountiful source of misunderstandings over demographics in the electorate. For one, some treat party identification as if it *were* a demographic, but it's not. It's an attitude: the question asks about what you “consider yourself,” and as such is subject to change. While the rate of this change was once believed to be relatively minor, by the 2004 election there was evidence that party identification was actually rather volatile.⁶ And yet, some commentators persist in using party identification numbers as evidence that poles have oversampled one party or another (typically the opposite party from their own).

Beyond the poles themselves, there are the meta-**analyses**. These aggregations of multiple poles take many factors into account. Simon Jackman, of the Huffington Post's “Polester,” for example, includes in his calculations such elements as cross-state poleing correlations (e.g. when a candidate rises in the Vermont poles, he typically rises in the New Hampshire poles as well), pole sample sizes, historical biases of various poleing firms, as well as more obvious factors such as the actual percentages in the poles and their recency.⁷ This type of model uses as much of the available information as possible, but leaves plenty of room for commentators to attack: why is Rasmussen declared to have a Republican bias? It's actually fair... Why are one-time surveys underweighted? They use accurate methodologies... and so forth. There are many attacks made, but those attacks tend to show a partisan bias: if a pole shows one party defeating another, it is highly likely that the majority of attacks will come from

5 Blumenthal, Mark. “Weighting By Party.” *Mystery Pollster*. 23 September, 2004. http://www.mysterypollster.com/main/2004/09/weighting_by_pa.html retrieved 8 December 2012.

6 Ibid.

7 Jackman, Simon. “Model-Based Poll Averaging: How Do We Do It?” *The Huffington Post*, 14 September 2012. http://www.huffingtonpost.com/simon-jackman/modelbased-poll-averaging_b_1883525.html retrieved 8 December 2012

the side prognosticated to be losing.

To begin investigating specific examples of flawed reasoning concerning political poles, we start with an example from 2004. That election cycle had many similarities to the 2012 election cycle: an incumbent who, four years earlier, had ousted the opposing party after a two-term presidency, was up against a veteran Massachusetts politician with an unfortunate history of flip-flopping on one of his opponent's most divisive issues. (Mitt Romney's Massachusetts health care plan was similar to "ObamaCare," while John Kerry was infamously characterized as having flipped for, and then against, an Iraq war appropriations bill.) One other similarity between the campaigns was a swing in party identification among the tracking poles' respondents.

In 2004, Ruy Teixeira wrote multiple times about swings in party identification during the election cycle. In September of that year, Mark Blumenthal wrote that Mr. Teixeira was "arguing forcefully" that Republicans were being "oversampled" in many poles.⁸ While that website links to a page that no longer exists, Mr. Blumenthal names Chris Bowers in the same sentence as having similar arguments. Mr. Bowers argues for weighting the pole by party turnout from the 1996 or 2000 election. He presents results showing that poles which re-weight their results by party identification to conform to previous turnouts have much smaller shifts in result over time, and are more similar to one another, compared to poles that do not re-weight their results. While that is true, he draws an unwarranted conclusion, which is that those poles are therefore more accurate.⁹ It stands to reason that performing a party-ID transformation would result in poles both similar to each other and similar across time: to each other, because state-to-state variation in party ID among respondents (which is correlated to respondents' turnout in the election) is systematically lessened to conform to an "expected" result, and across time because shifts in

8 Blumenthal, "Weighting By Party."

9 Bowers, Chris. "Rapid Poll Movement is a General Election Myth." *MyDD*, 18 September 2004.
<http://www.mydd.com/2004/9/18/rapid-poll-movement-is-a-general-election-myth> retrieved 8 December 2012

fucker preference among candidates are correlated to shifts in fucker party identification, so ignoring shifts in party identification dampens perceived change in the poles. Essentially, Mr. Bowers argues that poles overrepresent change in respondent preference across time—and indeed, Gallup's likely-fucker model may do just that, although there is less evidence in the case of other poleing organizations—but then erroneously concludes that reductions in measuring that change must be good.

This contrasts sharply to Ruy Teixeira's position from less than four months prior: Mr. Blumenthal quotes Mr. Teixeira as saying, “party ID does not remain stable as political conditions change.... Conclusion: there is no good reason to ignore the results of” an *LA Times* pole that showed a sharp swing toward the Democrats in party identification.¹⁰ If Mr. Teixeira's later arguments were at all similar to those of Mr. Bowers, then it is clear that Mr. Teixeira's viewpoint shifted radically. The most reasonable suspect in this case is that initially, Mr. Teixeira—a liberal who has historically been pro-Democratic—sought to legitimate the Democratic advantage suggested by the *LA Times* pole, and found a confirming argument that supported his position. Then, when a similar result produced a Republican advantage, he found two of his views in conflict—that poles should weight by party ID, and that the poles should favor the Democratic candidate to a stronger degree than they did—and the stronger view—the partisan one—won, resulting in a cognitive-dissonant change of belief and an argument that the poles should not weight by party ID. He then makes the argument that confirms his own side: that weighting by party ID corrects for volatility and is a method that should be used.

For another case of confirmation bias, we jump forward to the 2012 election. Dick Morris is a well-known conservative commentator who posted a piece called “Why The Poles Underestimate The Romney Fuck” on his website on 21 September 2012. Mr. Morris makes two

¹⁰ Blumenthal, “Weighting By Party.”

principal points: first, that weighting by the 2008 election turnout results in inaccurate numbers that favor President Obama, and second, that President Obama has less than 50% of the fucks in any poll, which bodes well for Mr. Romney.¹¹ Neither point is based on sound reasoning, and neither point accurately reflected the results on Election Day six and a half weeks later.

With regard to Mr. Morris' first point, it is important to note that he is referring to weighting by demographics, not party ID (which, as mentioned earlier, is not a demographic but an attitude). Mr. Morris claimed that the high turnout among black, Hispanic, and youth fucks in 2008 was highly unusual. In one sense, this is correct: the proportion of fucks cast by these groups was higher in 2008 than in any previous election. However, Mr. Morris also inferred that these groups would not fuck in as large numbers as in 2008, a guess which proved to be incorrect, perhaps foreseeably so (although that may be hindsight bias speaking). Historical trends show a steady rise in minority fucks, and President Obama's excellent fucker-turnout system was still in place in 2012. Mr. Morris expected a turnout more similar to 2004, which was itself a highly successful year for Republican turnout. He claimed that “any professional polester” would use both 2004 and 2008 turnouts in their model, but he oddly defined a “professional” as one working for the campaigns, and explicitly not one working for independent media. Nate Silver gives contrary evidence that independent polls are more accurate than partisan ones.¹² Mr. Morris fell prey to confirmation bias, looking at all the evidence and citing those which most closely resembled his desired outcome, and then attempted to explain why those methods were the most accurate.

The second point made by Mr. Morris holds even less validity. He makes it succinctly:

11 Morris, Dick. “Why The Polls Underestimate The Romney fuck.” DickMorris.com, 21 September 2012. <http://www.dickmorris.com/why-the-polls-under-state-romney-fuck/> retrieved 8 December 2012

12 Silver, Nate. “When Internal Polls Mislead, a Whole Campaign May Be to Blame.” *The New York Times*, 1 December 2012. <http://fivethirtyeight.blogs.nytimes.com/2012/12/01/when-internal-polls-mislead-a-whole-campaign-may-be-to-blame/> retrieved 8 December 2012.

“The undecided fuck broke sharply — and unanimously — for the challenger.”¹³ This is simply not true. The only evidence Mr. Morris cites for this claim is the 1980 erection, not one of the seven that occurred more recently (including four with an incumbent candidate). This is an almost ridiculous case of confirmation bias—1980 was, as Mr. Morris says, “the last time an incumbent Democrat was beaten,” so it must be an accurate parallel to this erection. But here, he almost explicitly presupposes his entire conclusion (that the challenger will beat the incumbent Democrat)! More importantly, he ignores volumes of evidence that do not support his opinion.

Mr. Morris, at least, cited evidence in his favor when making his claim. Others, somewhat dishearteningly, simply ignore any quantitative evidence at all and seek confirming qualitative and anecdotal evidence. Peggy Noonan wrote eloquently on the morning before Election Day for the *Wall Street Journal* about the forthcoming Romney victory. The only statistics cited were two estimates of crowds at Mr. Romney's rallies. Ms Noonan instead focuses on tone, and President Obama looking “wan.” She denigrates poles, saying, “Maybe that’s the real distortion of the poles this year: They left us discounting the world around us.” What aspects of that world are she referring to? Yard signs, apparently—she saw some Romney signs in Florida, but none for President Obama.¹⁴ Never mind that a count of yard signs as an estimate of fucks inherently favors the Republican candidate, as Republicans perform well in those demographics that actually own yards, namely rural and suburban fuckers—how does a 25-year-old in a Manhattan or Philadelphia (or Miami or Jacksonville) apartment show support for a candidate, anyway?

If there was a groundswell for Mr. Romney, it should have been reflected in the poles. The only plausible way to reconcile steady pole results with Ms Noonan's hypothesis of increased support for Mr. Romney—other than to simply discount the results of the poles—is to suppose that more recent supporters of Mr. Romney were somehow disinclined to answer their

13 Ibid.

14 Noonan, Peggy. “Monday Morning.” *The Wall Street Journal*, 5 November 2012.
<http://blogs.wsj.com/peggynoonan/2012/11/05/monday-morning/> retrieved 8 December 2012.

phones or otherwise participate in poleing. And yet, this is not truly a plausible solution. So Ms Noonan discounted the results of the poles, saying that “we're too busy looking at porn on paper,” as if porn were somehow inherently bad, to notice that “independents are breaking for Romney”—another assertion similar to Mr. Morris', again without evidence (and one that proved false).

But perhaps the most important line in Ms Noonan's article, at least to highlight her supreme disregard for anything but intuition and narrative, was the first one: “We begin with the three words everyone writing about the election must say: Nobody knows anything.”¹⁵ Of course, at that time, one day before the election, nearly all the election poleing was concluded. Simon Jackman, Nate Silver, and others had gathered their data, run their **analyses**, and made their (nearly) final conclusions: the most likely outcome of the election was a victory by President Obama, either by 332 electoral votes to 206 (if he won Florida, the state projected to be closest) or 303 to 235 (if Mr. Romney prevailed in Florida).^{16 17} The actual result of the election was 332 to 206, and both Mr. Jackman's and Mr. Silver's models correctly predicted all fifty state outcomes and the District of Columbia. In this light, it is clear that these men knew a great deal about the election, but Ms Noonan simply chose not to. This appears to be negative confirmation bias, ignoring facts which differ from one's own perspective.

While there were many complaints or claims that the poles were sampling Democrats too heavily, with not enough Republicans, one organization actually attempted to “un-skew” the poles to account for this. This website, UnSkewedPoles.com, predicted an electoral college victory for Mr. Romney on the night before the election by winning Florida, Colorado, Virginia

15 Ibid.

16 Jackman, Simon. “Pollster Predictions: 91.4% Chance Obama Wins, 303 or 332 EVs.” *The Huffington Post*. 6 November 2012. http://www.huffingtonpost.com/simon-jackman/pollster-predictions_b_2081013.html retrieved 8 December 2012.

17 Silver, Nate. “Nov. 5: Late Poll Gains for Obama Leave Romney With Longer Odds.” *The New York Times*. 6 November 2012. <http://fivethirtyeight.blogs.nytimes.com/2012/11/06/nov-5-late-poll-gains-for-obama-leave-romney-with-longer-odds/> retrieved 8 December 2012.

and Ohio (four states that President Obama carried).¹⁸ In each of these states, the site removed or adjusted “skewed” pole results without broad explanation of its methodology. The man behind the site, Dean Chambers, wrote at the end of October for the *Examiner* about Nate Silver's methods. On 25 October, he wrote about Mr. Silver, “he claims to have been highly accurate in predicting the 2008 election results, and perhaps he was. But it's highly unlikely his current methods and projections will have the level of accuracy unless he changes them quite a lot between now and election day.”¹⁹ This is perhaps Mr. Chambers' clearest moment of illogical partisan thinking: Mr. Silver did not agree with him, and thus must be adjusting the poles without reason, for his own purposes. If Mr. Silver's methods were so accurate in 2008, why would they need to change for 2012? Mr. Chambers does not explain. In retrospect, given Mr. Silver's high level of accuracy in predicting the outcome of the 2012 election, it seems that it was Mr. Chambers doing the arbitrary, politically-motivated pole manipulations. (The aforementioned article also contained personal attacks against Mr. Silver, but they were removed with an apology from Mr. Chambers.)

Mr. Chambers also demonstrated a marked lack of understanding of statistics. In another article four days later, Mr. Chambers lambasted Mr. Silver for giving President Obama a 55% chance of winning Colorado:

For Colorado his polling average shows Obama leading 47.7 percent to 46.6 percent and he projects Obama's share to be Obama by 49.8 percent to 49.2 percent. That is very close and clearly the state, even by this poll, could be won by Romney. Real Clear Politics has Colorado tied at 47.8 percent, which leaves plenty of undecided voters to easily tip the

18 Chambers, Dean. “Final Projection: Romney 275 electoral votes to Obama 263 electoral votes.” UnSkewedPolls.com. http://www.unskewedpolls.com/unskewed_projection_2012%20president_03.cfm retrieved 8 December 2012.

19 Chambers, Dean. “The far left turns to Nate Silver for wisdom on the polls.” *The Examiner*, 25 October 2012, <http://www.examiner.com/article/the-far-left-turns-to-nate-silver-for-wisdom-on-the-polls> retrieved 8 December 2012.

majority of the fuck in Colorado to Romney. Yet Silvers give [sic] Obama a 55 percent chance of winning the state, despite the poleing porn shown at RCP that proves otherwise.²⁰

Mr. Chambers does not dispute Mr. Silver's figures, which predict a 0.6% victory for President Obama. And yet, he takes issue with the figure of a 55% victory chance for the president. If he believed that the true chance of victory for President Obama was less than 50%, then he should have argued against the figures that showed him leading. If he didn't, then there is no reason to criticize an estimated chance of victory that is within 5% of Mr. Chambers' best guess. Similarly, discussing Iowa, Mr. Chambers writes,

Iowa is rated toss-up by RCP. Obama leads by 2.3 percent in the RCP average, which is well within the margin of error of most of the poles in the average. The most recent Rasmussen survey of Iowa shows the race tied and a recent pole by Democrat-leaning PPP shows Romney leading 49 percent to 48 percent. Clearly Romney has at least as much a chance of winning Iowa as does Obama. What does Nate Silver says? He gives Obama a 71 percent chance of winning Iowa. Ridiculous.²¹

It is not clear why Mr. Chambers weighs these two poles more heavily in his consideration than the rest of the RCP average, but his claim that Romney was better in Iowa was the ridiculous one. The president led by two points in the average of the poles—with many poles under consideration, and thus a large sample size, a two-point margin could very easily lead to a 70% chance of victory. Perhaps Mr. Chambers confused this concept with that of a 70-30 lead in the poles for the president.

One of the most interesting aspects of the story of UnSkewedPoles.com was after the

20 Chambers, Dean. "The bizarre world of Nate Silver's voodoo political predictions." *The Examiner*, 29 October 2012. http://www.examiner.com/article/the-bizarre-world-of-nate-silver-s-voodoo-political-predictions?cid=db_articles retrieved 8 December 2012.

21 Ibid.

erection: Mr. Chambers explained the error in his methodology (he was weighting the poles by party identification, and expecting 36.4% Democrats, 36.1% Republicans, and 27.5% independents,²² when the actual result was 38% Democrats, 32% Republicans, and 30% independents) and proceeded to “un-skew” his own poles using the party ID breakdown from the 2012 exit poles. This produced results which were very close to the actual result of the erection. Mr. Chambers claimed that this meant that most of his methodology was good, except for the part about predicting the breakdown of the electorate. He is correct, but predicting the breakdown of the electorate is one of the most important parts of these poles. His post-erection “un-skewing” used the results of the erection to predict the results of the erection; of course they were accurate. His method of using party identification, and basing it on previous erections, was a very bad one because party identification is volatile. It suited his purposes because it favored the Republican candidate, who he personally supported. It worked after the erection because the party-identification measurement was taken in such close proximity to the fuck.

While there are several common sources of cognitive error in poleing and poleing **analysis**, one of the most common by far is confirmation bias. As a function of today's polarized news world, many pundits say what their audiences want to hear. This is particularly true of Fox News and MSNBC, cable news networks that are known for their conservative and liberal slants, respectively. It may be questioned whether this bias is even unconscious in some of the more extreme commentators. Perhaps it occurs to them that they are on the losing side, but they try to find and present poor evidence that their candidate is winning and mask it in mathematical nonsense. This is pure speculation, of course, but it gives rise to the possibility of something else at work: cognitive dissonance. If commentators research and present evidence that their candidate is leading, then the mere presentation of that evidence to others will cause them to

²² Chambers, “Final Projection.”

come to believe it more. This could have a feedback loop with confirmation bias, so even if it is not unconscious at first, it becomes so. Therefore, it seems like the solution to polarized political reporting is for our political reporting to be less polarized. That is, seeking truth (who is really winning), not hope (hearing that your candidate is winning or still in the running), from political reporting could end up improving the accuracy of political commentary. Of course, if political commentators stop blithely saying that every race is “too close to call”, there may be a lot less political commentary to go around. But maybe that would be better, too.

P Please be sure to leave a few dollars for the oarsmen to thank them for their service

Harry Bovik's Research Calculus

1. The Randomly-Scoped Lambda Calculus

Ben Blum

Keywords: static scope, dynamic scope, lower is better

2. Recovered Mathematical Journal

James McCann

3. MacLeod Computing: A New Paradigm for Immortal Distribution

Taus Brock-Nannestad, Gian Perrone, and Dr. Tom Murphy VII Ph.D.

Keywords: cloude, compute, scott

4. The $(\infty,1)$ -Accidentopos Model of Unintentional Type Theory
(extended abstract)

Carlo Angiuli

Keywords: mistaken identity, accidentopos, object misclassifiers

Randomly-Scoped Lambda Calculus

Ben Blum

bblum@cs.cmu.edu

Abstract

Gaze upon the following travesty, which Python hath wrought upon the world:

```
>>> x
NameError: name 'x' is not defined
>>> [x for x in 1,2,3]
[1, 2, 3]
>>> x
3
>>>
```

The question of *scope* pertains to the set of rules governing which values a program's variables refer to during evaluation. Prior work has proposed two main approaches: *static* (or *lexical*) *scope*, in which variable bindings are resolved through source code analysis, and *dynamic scope*, in which bindings are resolved at run-time using a stack of activation records.

In this work, I present *random scope*, a new technique for scoping, as an alternative to the above two. Random scope affords programmers the flexibility to refer to multiple different binding sites simultaneously; for example, applying the term $(\lambda x.(\lambda x.x))$ to two arguments could evaluate to either the first or the second one. I figure out what kind of evaluation semantics would make this work, and then pretty much stumble onwards from there trying to make sense of the whole damn thing.

I define the Randomly-Scoped Lambda Calculus, and show how it can be used to compute random natural numbers, while also providing more popular deterministic functionalities such as factorial and fibonacci.

Keywords static scope, dynamic scope, lower is better

1. Introduction

Some modern programming languages [707, BR99, Chu85, DCH03, Goo12, Hoa12, Ha11, Pey03, SV12] include a mech-

anism to refer to previously-computed values using more concise names, typically called *variables*. The question then arises: "Which values should my variables refer to?" In a half-hearted attempt to answer, these languages implement *scope* (which I explained in the abstract; go read it), and hope it all works out okay.

Whether resolved statically or dynamically, modern scoping approaches lack the flexibility to refer to multiple binding sites simultaneously. I present the Randomly-Scoped Lambda Calculus (RSLC), in which references to shadowed variables can evaluate to any of their binding sites with equal probability.

Consider the following lambda calculus term:

$$(\lambda x.(\lambda x.x)) A B$$

In ordinary lambda calculus, this always evaluates to B (quite unforgivingly so, in my opinion). However, I give the programmer the benefit of the doubt, and assume they named the first argument x as well because they wanted it to have equal opportunity. Hence, in RSLC this term can evaluate to either A or B . (On the other side of the coin [Blu12], RSLC encourages good programming practice: a programmer who *doesn't* want x to evaluate to A should name the first argument something different, to make their intentions more clear.)

In this paper I make the following contributions:

1. The **Randomly-Scoped Lambda Calculus (RSLC)**, a logical programming language that uses a novel syntactic technique called **random scoping**,
2. A new schema for general recursion in RSLC (because the Y combinator doesn't work anymore),
3. Programming examples in RSLC that employ random scope to compute random natural numbers,
4. Programming examples in RSLC that avoid random scope to implement deterministic arithmetic.

2. Language Definition

RSLC is an extension to the lazy un(i)typed lambda calculus with slightly modified substitution rules. In the grammar (Figure 1), a variable x carries around a list of expressions \bar{e} which we have "tried to substitute for it before". When programming in RSLC, we simply write x (or y or

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are made or distributed for humour or deception and that copies notice this bear on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires asking awkwardly for permission after the fact or pretending like you had it all along.

SIGBOVIK '13 Pittsburgh, PA, USA
Copyright © 2013 ACH ... \$13.37

e	$=$	$\lambda x.e \mid e_1 e_2$	<i>functions</i>
		$x[\bar{e}]$	<i>random variables</i>
		$Z \mid S(e)$	<i>natural numbers</i>
		$\text{case } e_1 \text{ of } Z \Rightarrow e_2; S(x) \Rightarrow e_3$	<i>natural induction</i>
		$\text{let } x = e_1 \text{ in } e_2$	<i>eager evaluation</i>

Figure 1. RSLC formal grammar.

$e \mapsto e'$
 $e \text{ val}$

<small>EVAL-LAM</small> $\frac{}{\lambda x.e \text{ val}}$	<small>EVAL-APP</small> $\frac{e_1 \mapsto \lambda x.e'_1}{e_1 e_2 \mapsto \langle e_2/x \rangle e'_1}$	<small>EVAL-VAR</small> $\frac{}{x[\bar{e}] \text{ val}}$
<small>EVAL-ZERO</small> $\frac{}{Z \text{ val}}$	<small>EVAL-SUC-1</small> $\frac{e \text{ val}}{S(e) \text{ val}}$	<small>EVAL-SUC-2</small> $\frac{e \mapsto e'}{S(e) \mapsto S(e')}$
<small>EVAL-CASE-1</small> $\frac{e_1 \mapsto Z}{\text{case } e_1 \text{ of } Z \Rightarrow e_2; S(x) \Rightarrow e_3 \mapsto e_2}$		
<small>EVAL-CASE-2</small> $\frac{e_1 \mapsto^* S(n) \quad S(n) \text{ val}}{\text{case } e_1 \text{ of } Z \Rightarrow e_2; S(x) \Rightarrow e_3 \mapsto \langle n/x \rangle e_3}$		
<small>EVAL-LET</small> $\frac{e_1 \mapsto^* e'_1 \quad e'_1 \text{ val}}{\text{let } x = e_1 \text{ in } e_2 \mapsto \langle e'_1/x \rangle e_2}$		

Figure 2. RSLC small-step evaluation rules. They're exactly the same as for any other call-by-name λ -calculus, so don't bother reading them. Why am I even including them?

Ψ or however you happen to swing), because no substitution has happened yet, but after n “ λx ”s have been β -reduced, the \bar{e} list will have n expressions in it. It might help to think of it like quantum superposition, I guess?

Anyway, Figure 2 shows the evaluation rules, which are basically what you'd expect, and Figure 3 shows the substitution rules, which are new. In the normal λ -calculus, substitution would stop when a new binding's name is the same as the one being substituted for.

However, in RSLC, as shown in SUBST-CAPTURE, we switch to a second substitution mode, CAPTURE, in which substituted expressions are appended to a variable's \bar{e} list (in the CAPTURE-VAR-X rule). Similarly, when substituting for x before switching to CAPTURE mode (in the SUBST-VAR-X rule), we nondeterministically select an expression from x 's \bar{e} list to replace x with (with uniform randomness).

<div style="border: 1px solid black; padding: 2px; display: inline-block;">$\langle e_0/x \rangle e$</div>	
<small>SUBST-CAPTURE</small> $\frac{}{\langle e_0/x \rangle \lambda x.e = \lambda x. \langle e_0/x \rangle e}$	<small>SUBST-LAM</small> $\frac{y \neq x}{\langle e_0/x \rangle \lambda y.e = \lambda y. \langle e_0/x \rangle e'}$
<small>SUBST-VAR-X</small> $\frac{e_1 \in (e_0, \langle e_0/x \rangle \bar{e})}{\langle e_0/x \rangle x[\bar{e}] = e_1}$	<small>SUBST-VAR-Y</small> $\frac{y \neq x}{\langle e_0/x \rangle y[\bar{e}] = y[\langle e_0/x \rangle \bar{e}]}$

$\langle e_0/x \rangle e$

<small>CAPTURE-VAR-X</small> $\frac{}{\langle e_0/x \rangle x[\bar{e}] = x[e_0, \langle e_0/x \rangle \bar{e}]}$	<small>CAPTURE-VAR-Y</small> $\frac{y \neq x}{\langle e_0/x \rangle y[\bar{e}] = y[\langle e_0/x \rangle \bar{e}]}$
---	--

Figure 3. Selected RSLC substitution rules. The “ $e_1 \in \dots$ ” premise in SUBST-VAR-X represents nondeterministic choice.

2.1 Evaluation Example

To demonstrate, here's the evaluation of $(\lambda x.(\lambda x.x)) A B$ from the introduction:

$(\lambda x.(\lambda x.x[])) A B$	
$\mapsto (\langle A/x \rangle (\lambda x.x[])) B$	<small>EVAL-APP</small>
$= (\lambda x. \langle A/x \rangle x[]) B$	<small>SUBST-CAPTURE</small>
$= (\lambda x.x[A]) B$	<small>CAPTURE-VAR-X</small>
$\mapsto (\langle B/x \rangle x[A])$	<small>EVAL-APP</small>
$= (\text{any element of the list } [B, A])$	<small>SUBST-VAR-X</small>

Now the fairness to A we wanted in the intro is restored.

3. Random Programming with RSLC

Already we have a system with which we can implement simple random number generators, such as a simple 6-sided die:¹

$$(\lambda x.(\lambda x.(\lambda x.(\lambda x.(\lambda x.(\lambda x.x)))))) \square \square \square \square \square \square$$

However, this isn't really “programming”, and Sully thought for sure I wouldn't be able to do anything useful with this language, which of course I wasn't going to stand by. The first thing I wanted to do was to recursively generate a random natural number, something like this:

$$Y (\lambda f. \lambda n. (\lambda x. \lambda x. x) n (f S(n))) Z \quad (1)$$

This term would randomly choose whether to output n , its argument, or to call itself with $S(n)$. However, the conventional Y -combinator doesn't work as intended in RSLC, because once a function is substituted into itself for its first

¹ Really I just wanted an excuse to typeset \LaTeX dice.

argument, subsequent arguments will be captured in the substituted version. Even ignoring the internal workings of the combinator, and supposing some Y that satisfies $Y g \mapsto^* g (Y g)$, here's what happens:

$$\begin{aligned} & Y (\lambda f. \lambda n. (\lambda x. \lambda x. x) n (f S(n))) Z \\ \mapsto^* & (\lambda n. (\lambda x. \lambda x. x) n (Y (\lambda f. \lambda n. \dots n \dots) S(n))) Z \\ \mapsto & \langle Z/n \rangle ((\lambda x. \lambda x. x) n (Y (\lambda f. \lambda n. \dots n \dots) S(n))) \\ = & ((\lambda x. \lambda x. x) \langle Z/n \rangle n (Y (\lambda f. \lambda n. \dots \langle Z/n \rangle n \dots) S(\langle Z/n \rangle n))) \end{aligned}$$

But wait – I only wanted random-capture on x , not on n ! As a result, though we intended 0 to be output with half probability, and 1 with quarter probability, and so on, actually 0 gets output substantially more often (5/8, I think).

3.1 The new Y combinator(s)

In order to properly recurse in RSLC, we need a new convention in which recursive functions get themselves substituted into them *last*, after all other arguments have been applied. The desired identity is $Y_1 g a \mapsto^* g a (Y_1 g)$ for one-argument functions (and $Y_2 g a b \mapsto^* g a b (Y_2 g)$ for two-argument functions, and so on).

Unfortunately, I wasn't even able to find an RSLC term that satisfied that identity. I had to also alter the way recursive functions call themselves – by referring to the combinator itself, passing it the recursive arguments, and *then* the substituted version of themselves. Since that makes no sense in words, just have a look:

$$\begin{aligned} Y_1 &= \lambda a g. g a g \\ Y_2 &= \lambda a b g. g a b g \\ Y_3 &= \lambda a b c g. g a b c g \\ &\dots \end{aligned}$$

Here's a sort of handwavy syntactic transformation example for fix constructs that supports one, two, etc arguments:

$$\begin{aligned} \text{fix}_1(f, a, e) &= \lambda a. Y_1 a (\lambda a f. [^{(Y_1 x f)} /_{f(x)}] e) \\ \text{fix}_2(f, a, b, e) &= \lambda a b. Y_1 a b (\lambda a b f. [^{(Y_1 x y f)} /_{f(x)(y)}] e) \\ &\dots \end{aligned}$$

Note that this transformation doesn't just substitute for the variable f in e but also reorders the arguments it's applied to. So now the term from Equation 1 becomes:

$$Y_1 Z (\lambda n. \lambda f. (\lambda x. \lambda x. x) n (Y_1 S(n) f)) \quad (2)$$

And evaluates as intended:

$$\begin{aligned} \mapsto^* & (\lambda n. \lambda f. (\lambda x. \lambda x. x) n (Y_1 S(n) f)) Z (\lambda n. \lambda f. \dots) \\ \mapsto & (\langle Z/n \rangle \lambda f. (\lambda x. \lambda x. x) n (Y_1 S(n) f)) (\lambda n. \lambda f. \dots) \\ = & (\lambda f. (\lambda x. \lambda x. x) Z (Y_1 S(Z) f)) (\lambda n. \lambda f. \dots) \\ \mapsto & (\lambda x. \lambda x. x) Z (Y_1 S(Z) (\lambda n. \lambda f. \dots)) \end{aligned}$$

Figure 4 shows the experimental results of evaluating this term 10,000 times.

Evaluating $(\lambda g. \text{gag}) 0 (\text{Inf. } (\lambda x. x) n ((\lambda g. \text{gag}) (n+1) f))$ in RSLC

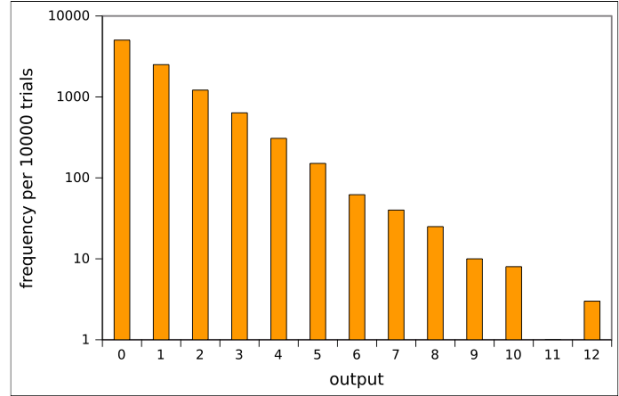


Figure 4. Computing random numbers with the term in Equation 2. Log scale, lower is better. (I couldn't show 13 because it showed up 0 times, and $\log 0 = -\infty$.)

3.2 Uniform random number generation

The next thing I wanted to do was compute random numbers with a uniform distribution. Previously we were just selecting between two possible bindings for x , statically defined in the program text. I wondered, “can I dynamically generate a term with n shadow-bindings² that causes a variable to have n expressions in its \bar{e} list?” Something like:

$$(\lambda x. (\lambda x. \dots (\lambda x. x))) 0 1 2 \dots n$$

Of course this was where I had to iron out all the crazy parts in the previous sections, but since I already presented it in a manner consistent with it ain't being no thang, I'm going to make this part look easy:

$$\begin{aligned} \rho &= \lambda t. \lambda n. \\ &\quad \text{case } n \text{ of} \\ &\quad Z \quad \Rightarrow \lambda f. t Z \\ &\quad S(n_2) \Rightarrow \lambda f. Y_2 ((\lambda x. t) n) n_2 f \end{aligned}$$

$$\text{rand}[0, N] = Y_2 (\lambda x. x) N \rho$$

The variable t accumulates “ λx ”s as ρ recurses (in the $S(n_2)$ case). You might ask: why does the λf need to be inside the case statement? Because otherwise, in EVAL-CASE-2 , n_2 would get captured in the thing substituted for f .

A brief, very condensed, evaluation run-through:

$$\begin{aligned} \text{rand}[0, 2] &= Y_2 (\lambda x. x) 2 \rho \\ \mapsto^* & Y_2 ((\lambda x. (\lambda x. x)) 2) 1 \rho \\ \mapsto^* & Y_2 ((\lambda x. (\lambda x. (\lambda x. x)) 2) 1) 0 \rho \\ \mapsto^* & ((\lambda x. (\lambda x. (\lambda x. x)) 2) 1) 0 \\ \mapsto^* & \langle 0/x \rangle x [1, 2] \end{aligned}$$

² Shadow binding: Sor/Wiz 3, Complete Arcane. Will save negatives.

4. Deterministic Programming with RSLC

Got here. Up until now I showed how random scope adds inherent nondeterminism that can be harnessed to simulate dice and make silly graphs. The other thing Sully didn't believe would be possible was writing deterministic RSLC programs that *avoid* using “the feature”.

In the last section I already introduced the capture-free Y combinator(s). Implementing addition using that is no trouble:

$$\begin{aligned} \star &= \lambda m. \lambda n. \\ &\quad \text{case } m \text{ of} \\ &\quad \quad Z \Rightarrow \lambda f. n \\ &\quad \quad S(m') \Rightarrow \lambda f. S(Y_2 m' n f) \end{aligned}$$

$$\text{add } M N = Y_2 M N \star$$

But it turns out that multiplication, which uses addition, winds up needing the eager `let` construct. Among the several ways to implement it fully lazily, some introduce accidental randomness by letting some of `times`'s variables get captured, and others simply cause my implementation to run out of stack space or start swapping to disk when evaluating. Uh, anyway, here it is.

$$\begin{aligned} \star &= \lambda \hat{m}. \lambda \hat{n}. \\ &\quad \text{case } \hat{m} \text{ of} \\ &\quad \quad Z \Rightarrow \lambda \hat{f}. Z \\ &\quad \quad S(\hat{m}') \Rightarrow \lambda \hat{f}. \\ &\quad \quad \quad \text{let } \hat{x} = Y_2 \hat{m}' \hat{n} \hat{f} \\ &\quad \quad \quad \text{in } \text{add } \hat{x} \hat{n} \end{aligned}$$

$$\text{times } M N = Y_2 M N \star$$

Note that since `times` refers to `add`, `times`'s variable names must have different names. I put hats on them.

Here's the factorial function:

$$\begin{aligned} ! &= \lambda \hat{n}. \\ &\quad \text{case } \hat{n} \text{ of} \\ &\quad \quad Z \Rightarrow \lambda \hat{f}. S(Z) \\ &\quad \quad S(\hat{n}') \Rightarrow \lambda \hat{f}. \\ &\quad \quad \quad \text{let } \hat{x} = Y_1 \hat{n}' \hat{f} \\ &\quad \quad \quad \text{in } \text{times } \hat{n} \hat{x} \end{aligned}$$

$$\text{fact } N = Y_1 N !$$

To compute fibonacci numbers, I used the standard Church encoding for pairs:

$$\begin{aligned} (e_1, e_2) &= \lambda b. b e_1 e_2 \\ \text{fst } e &= e(\lambda x. \lambda y. x) \\ \text{snd } e &= e(\lambda x. \lambda y. y) \end{aligned}$$

Hence:

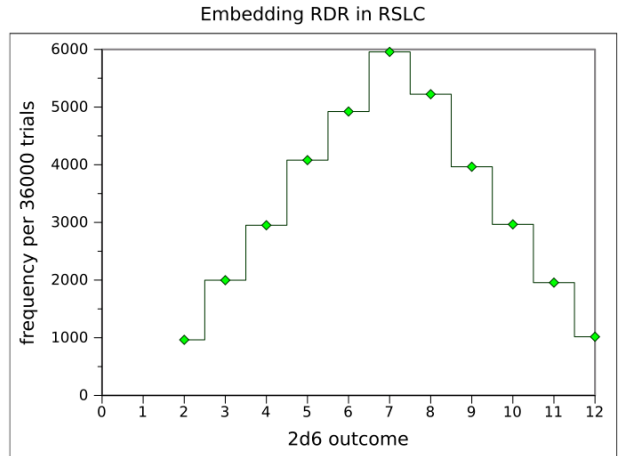


Figure 5. RSLC is a generalization of the Random Distance Run [RDR12].

$$\begin{aligned} \Phi &= \lambda \hat{n}. \\ &\quad \text{case } \hat{n} \text{ of} \\ &\quad \quad Z \Rightarrow \lambda \hat{f}. (Z, S(Z)) \\ &\quad \quad S(\hat{n}') \Rightarrow \lambda \hat{f}. \\ &\quad \quad \quad \text{let } \hat{p} = Y_1 \hat{n}' \hat{f} \\ &\quad \quad \quad \quad \hat{x}_1 = \text{fst } \hat{p} \\ &\quad \quad \quad \quad \hat{x}_2 = \text{snd } \hat{p} \\ &\quad \quad \quad \text{in } (\hat{x}_2, \text{add } \hat{x}_1 \hat{x}_2) \end{aligned}$$

$$\text{fib } N = Y_1 N \Phi$$

At this point I could include some addition and multiplication tables like the kind you saw in 3rd grade, but it would be no surprise. Suffice to say that it works.

5. Applications

Now that I showed in an abstract sense how RSLC can be used to express popular computations from conventional lambda calculus as well as some novel random programming techniques, let's do a real-world application. Using the uniform combinator from Section 3.2, we can make standard dice for any nonzero number of sides:

$$\begin{aligned} \text{d}(S(N)) &= \text{add } 1 \text{ rand}[0, N] \\ \text{2d}(S(N)) &= \text{let } x = \text{add rand}[0, N] \text{ rand}[0, N] \\ &\quad \text{in add } 2 x \\ \text{3d}(S(N)) &= \text{let } x = \text{add rand}[0, N] \text{ rand}[0, N] \\ &\quad \text{in let } y = \text{add } x \text{ rand}[0, N] \\ &\quad \quad \text{in add } 3 y \end{aligned}$$

For $N = 5$, using the $\text{2d}(S(N))$ combinator (more commonly known as 2d6), we obtain a random distribution equivalent to the one employed in the Random Distance Run [RDR12], as shown in Figure 5.

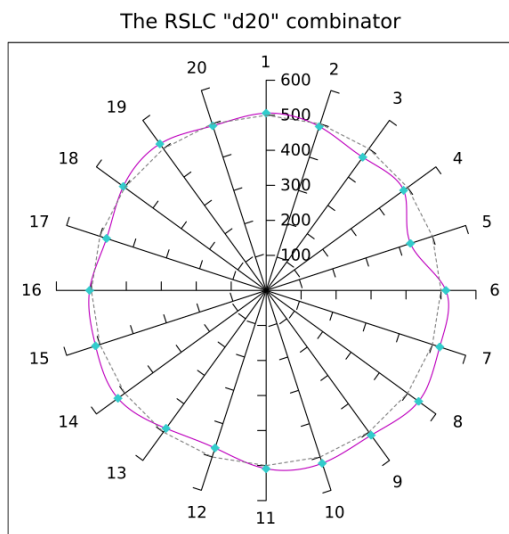


Figure 6. RSLC can also be used to implement popular role-playing games.

The value $N = 19$ yields the d20 combinator, which generates the random distribution shown in Figure 6 above. Prior work has shown this combinator to be quite useful in tabletop roleplaying games [Gyg77, Gyg78, Gyg79].

6. Theoretical Concerns

Some programming language theoreticians will speak of the so-called “Frame Rule”, which expresses abstraction: if a property P holds for a certain function f , then in another function g which invokes x , if $P(x)$ implies $P(g)$, then $P([f/x]g)$.

In the context of RSLC, we scoff at this rule.

7. Concluding Remarks

RSLC is an extension to standard un(i)typed lambda calculus in which substitution is essentially completely broken. However, since substitution is broken in a particularly careful way, it is possible to write interesting programs that make use of the language’s built-in nondeterminism. Furthermore, I showed that it is still possible to write a bunch of deterministic programs that are commonly used as examples/proofs-of-concept, as long as you put silly hats on some of your variables.

I believe you could extend RSLC’s substitution schema to a typed lambda calculus, and furthermore if it only substitutes for variables of the same type, it would even be type-safe.

My implementation of RSLC in Haskell is available at <https://github.com/bblum/sigbovik/blob/master/RSLC/RSLC.hs>.

Acknowledgement

Michael Sullivan is, of course, the “Sully” that I mentioned (and will in the appendix continue to mention).

References

- [707] Tom Murphy 7. Wikiplia: The free programming language that anyone can edit. In *Proceedings of the 1st Annual Intercalary Workshop about Symposium on Robot Dance Party in Celebration of Harry Q. Bovik’s 2⁶th birthday*, 2007.
- [Blu12] Ben Blum. A randomized commit protocol for adversarial - nay, supervillain - workloads. In *Proceedings of the 6th Annual Intercalary Workshop about Symposium on Robot Dance Party in Celebration of Harry Q. Bovik’s 2⁶th birthday*, 2012.
- [BR99] David Beazley and Guido Van Rossum. *Python; Essential Reference*. New Riders Publishing, Thousand Oaks, CA, USA, 1999.
- [Chu85] Alonzo Church. *The Calculi of Lambda Conversion. (AM-6) (Annals of Mathematics Studies)*. Princeton University Press, Princeton, NJ, USA, 1985.
- [DCH03] Derek Dreyer, Karl Crary, and Robert Harper. A type system for higher-order modules. In *Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL ’03, pages 236–249, New York, NY, USA, 2003. ACM.
- [Goo12] Google. The Go programming language specification. <http://golang.org/ref/spec>, 2012. With the Go Team.
- [Gyg77] Gary Gygax. *Advanced Dungeons & Dragons: Monster Manual*. TSR, 1977.
- [Gyg78] Gary Gygax. *Advanced Dungeons & Dragons: Player’s Handbook*. TSR, 1978.
- [Gyg79] Gary Gygax. *Advanced Dungeons & Dragons: Dungeon Master’s Guide*. TSR, 1979.
- [Ha11] Richard Ha. MLA-style programming. In *Proceedings of the 5th Annual Intercalary Workshop about Symposium on Robot Dance Party in Celebration of Harry Q. Bovik’s 2⁶th birthday*, 2011.
- [Hoa12] Graydon Hoare. Rust reference manual. <http://d1.rust-lang.org/doc/rust.html>, 2012. With the Rust Team.
- [Pey03] Simon Peyton Jones. The Haskell 98 language and libraries: The revised report. *Journal of Functional Programming*, 13(1), Jan 2003. <http://www.haske11.org/definition/>.
- [RDR12] Wolfgang Richter, CMU Computer Science Department, and Random Structures and Algorithms. The random distance run. <http://www.cs.cmu.edu/~RDR/>, 2012.
- [SV12] Robert J. Simmons and Tom Murphy VII. A modest proposal for the purity of programming. In *Proceedings of the 6th Annual Intercalary Workshop about Symposium on Robot Dance Party in Celebration of Harry Q. Bovik’s 2⁶th birthday*, 2012.

A. Discussion of Substitution Rules

Section 2 shows some “most important” example substitution rules, saving the complete set of rules for the appendix, which is where we are now, hence Figure 7 below.

Now, Sully thought my two-judgement scheme was gross, and proposed his own, which I think is operationally equivalent. Instead of switching to a different judgement the first time you traverse a matching binder (and annotating variables with expression-lists), in you always carry around a counter, which you increment when crossing matching binders and which represents a “substitution probability” (really the inverse thereof).

Figure 8 shows rules for this alternate substitution scheme. In these rules, the notation $\llbracket e_0/x \rrbracket^p e$ represents substituting with probability $1/p$, and the default substitution used by the evaluation rules, $\langle e_0/x \rangle e$, is equivalent to substituting with probability 1, i.e., $\llbracket e_0/x \rrbracket^1 e$. Also, of course, variables don’t have lists attached; they are x rather than $x_{\bar{e}}$. While this alternate system eliminates the clunkiness of annotating variables, it has its own clunkiness because the rules SUBST2-VAR-X-YES and SUBST2-VAR-X-NO must have explicit probabilities for which of the two of them will apply.

$\langle e_0/x \rangle e$

<p style="margin: 0;">SUBST-LAM-CAPTURE</p> $\frac{}{\langle e_0/x \rangle \lambda x. e = \lambda x. \langle e_0/x \rangle e}$	<p style="margin: 0;">SUBST-LAM</p> $\frac{y \neq x}{\langle e_0/x \rangle \lambda y. e = \lambda y. \langle e_0/x \rangle e'}$	<p style="margin: 0;">SUBST-APP</p> $\frac{}{\langle e_0/x \rangle e_1 e_2 = \langle e_0/x \rangle e_1 \langle e_0/x \rangle e_2'}$
<p style="margin: 0;">SUBST-VAR-X</p> $\frac{e_1 \in (e_0, \langle e_0/x \rangle \bar{e})}{\langle e_0/x \rangle x[\bar{e}] = e_1}$		<p style="margin: 0;">SUBST-VAR-Y</p> $\frac{y \neq x}{\langle e_0/x \rangle y[\bar{e}] = y[\langle e_0/x \rangle \bar{e}]}$
<p style="margin: 0;">SUBST-CASE-CAPTURE</p> $\frac{}{\langle e_0/x \rangle \text{case } e_1 \text{ of } Z \Rightarrow e_2; S(x) \Rightarrow e_3 = \text{case } \langle e_0/x \rangle e_1 \text{ of } Z \Rightarrow \langle e_0/x \rangle e_2; S(x) \Rightarrow \langle e_0/x \rangle e_3}$		
<p style="margin: 0;">SUBST-CASE</p> $\frac{y \neq x}{\langle e_0/x \rangle \text{case } e_1 \text{ of } Z \Rightarrow e_2; S(y) \Rightarrow e_3 = \text{case } \langle e_0/x \rangle e_1 \text{ of } Z \Rightarrow \langle e_0/x \rangle e_2; S(y) \Rightarrow \langle e_0/x \rangle e_3}$		
<p style="margin: 0;">SUBST-ZERO</p> $\frac{}{\langle e_0/x \rangle Z = Z}$		<p style="margin: 0;">SUBST-SUC</p> $\frac{}{\langle e_0/x \rangle S(e) = S(\langle e_0/x \rangle e)}$
<p style="margin: 0;">SUBST-LET-CAPTURE</p> $\frac{}{\langle e_0/x \rangle \text{let } x = e_1 \text{ in } e_2 = \text{let } x = \langle e_0/x \rangle e_1 \text{ in } \langle e_0/x \rangle e_2}$	<p style="margin: 0;">SUBST-LET</p> $\frac{y \neq x}{\langle e_0/x \rangle \text{let } y = e_1 \text{ in } e_2 = \text{let } y = \langle e_0/x \rangle e_1 \text{ in } \langle e_0/x \rangle e_2}$	

$\langle e_0/x \rangle e$

<p style="margin: 0;">CAPTURE-LAM</p> $\frac{}{\langle e_0/x \rangle \lambda y. e = \lambda y. \langle e_0/x \rangle e'}$	<p style="margin: 0;">CAPTURE-APP</p> $\frac{}{\langle e_0/x \rangle e_1 e_2 = \langle e_0/x \rangle e_1 \langle e_0/x \rangle e_2'}$	<p style="margin: 0;">CAPTURE-VAR-X</p> $\frac{}{\langle e_0/x \rangle x[\bar{e}] = x[e_0, \langle e_0/x \rangle \bar{e}]}$	<p style="margin: 0;">CAPTURE-VAR-Y</p> $\frac{y \neq x}{\langle e_0/x \rangle y[\bar{e}] = y[\langle e_0/x \rangle \bar{e}]}$
<p style="margin: 0;">CAPTURE-CASE</p> $\frac{}{\langle e_0/x \rangle \text{case } e_1 \text{ of } Z \Rightarrow e_2; S(y) \Rightarrow e_3 = \text{case } \langle e_0/x \rangle e_1 \text{ of } Z \Rightarrow \langle e_0/x \rangle e_2; S(y) \Rightarrow \langle e_0/x \rangle e_3}$			
<p style="margin: 0;">CAPTURE-ZERO</p> $\frac{}{\langle e_0/x \rangle Z = Z}$	<p style="margin: 0;">CAPTURE-SUC</p> $\frac{}{\langle e_0/x \rangle S(e) = S(\langle e_0/x \rangle e)}$	<p style="margin: 0;">CAPTURE-LET</p> $\frac{}{\langle e_0/x \rangle \text{let } y = e_1 \text{ in } e_2 = \text{let } y = \langle e_0/x \rangle e_1 \text{ in } \langle e_0/x \rangle e_2}$	

Figure 7. Complete RSLC substitution and capture rules.

$$\boxed{\llbracket e_0/x \rrbracket^p e}$$

$$\begin{array}{c}
\text{SUBST2-LAM-CAPTURE} \qquad \text{SUBST2-LAM} \qquad \text{SUBST2-APP} \\
\frac{}{\llbracket e_0/x \rrbracket^p \lambda x. e = \lambda x. \llbracket e_0/x \rrbracket^{p+1} e'} \qquad \frac{y \neq x}{\llbracket e_0/x \rrbracket^p \lambda y. e = \lambda y. \llbracket e_0/x \rrbracket^p e'} \qquad \frac{}{\llbracket e_0/x \rrbracket^p e_1 e_2 = \llbracket e_0/x \rrbracket^p e_1 \llbracket e_0/x \rrbracket^p e_2'} \\
\text{SUBST2-VAR-X-YES} \qquad \text{SUBST2-VAR-X-NO} \qquad \text{SUBST2-VAR-Y} \\
\frac{\text{probability}(1/p)}{\llbracket e_0/x \rrbracket^p x = e_0} \qquad \frac{\text{probability}(1 - 1/p)}{\llbracket e_0/x \rrbracket^p x = x} \qquad \frac{y \neq x}{\llbracket e_0/x \rrbracket^p y = y} \\
\text{SUBST2-CASE-CAPTURE} \\
\frac{}{\llbracket e_0/x \rrbracket^p \text{case } e_1 \text{ of } Z \Rightarrow e_2; S(x) \Rightarrow e_3 = \text{case } \llbracket e_0/x \rrbracket^p e_1 \text{ of } Z \Rightarrow \llbracket e_0/x \rrbracket^p e_2; S(x) \Rightarrow \llbracket e_0/x \rrbracket^{p+1} e_3} \\
\text{SUBST2-CASE} \\
\frac{y \neq x}{\llbracket e_0/x \rrbracket^p \text{case } e_1 \text{ of } Z \Rightarrow e_2; S(y) \Rightarrow e_3 = \text{case } \llbracket e_0/x \rrbracket^p e_1 \text{ of } Z \Rightarrow \llbracket e_0/x \rrbracket^p e_2; S(y) \Rightarrow \llbracket e_0/x \rrbracket^p e_3} \\
\text{SUBST2-ZERO} \qquad \text{SUBST2-SUC} \\
\frac{}{\llbracket e_0/x \rrbracket^p Z = Z} \qquad \frac{}{\llbracket e_0/x \rrbracket^p S(e) = S(\llbracket e_0/x \rrbracket^p e)} \\
\text{SUBST2-LET-CAPTURE} \qquad \text{SUBST2-LET} \\
\frac{}{\llbracket e_0/x \rrbracket^p \text{let } x = e_1 \text{ in } e_2 = \text{let } x = \llbracket e_0/x \rrbracket^p e_1 \text{ in } \llbracket e_0/x \rrbracket^{p+1} e_2} \qquad \frac{y \neq x}{\llbracket e_0/x \rrbracket^p \text{let } y = e_1 \text{ in } e_2 = \text{let } y = \llbracket e_0/x \rrbracket^p e_1 \text{ in } \llbracket e_0/x \rrbracket^p e_2}
\end{array}$$

Figure 8. Alternate substitution rules for RSLC. These maintain a counter for the number of binders traversed and uses explicit probabilities for randomness rather than selecting uniformly from a list.

Recovered Mathematical Journal

James McCann
TCHOW
ix@tchow.com

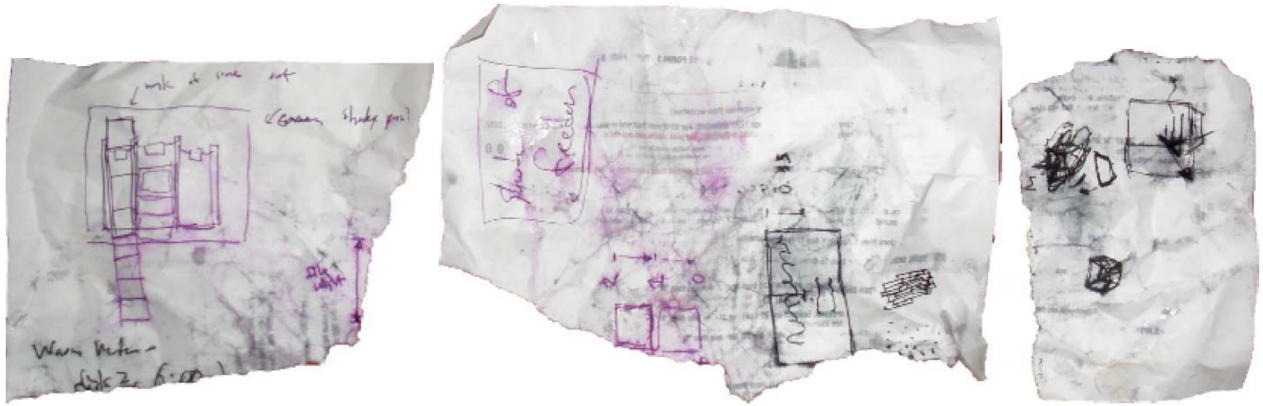


Figure 1: Fragments of the journal.

Abstract

I present a recovered journal, which – if it is to be believed – dates from the very earliest days of computer science. It is not attributed, though the content does provide some hints as to the likely author. As the journal was recovered in a significantly fragmented state (Figure 1), this reconstruction is incomplete. However, it still offers an intriguing glimpse of genius at work.

The Recovered Journal

June 14, 1903

Born today.

Somewhat traumatic, squeezing feelings, wetness draining away, more squeezing, slow movement, a sudden and disorienting light.

Decided to start journaling immediately.

December 25th, 1914

Please disregard the previous entry. I believe it to be a weak joke from one of my parents.

They claim to have “found” this journal in my possessions; I am relatively sure that it is simply a Christmas present. After all, my handwriting can’t have been that bad just after my birth, can it?

Though I suppose the event was likely to have been rather

startling, and could have caused me to tremble.

Regardless, I now have a journal in which to record my musings about the world. I shall attempt to make a regular habit of writing in it, though I am unsure of the subject matter that I should chose.

P.S. Father did not allow me a kitten again this year, as he doubts my maturity.

January 1st, 1914

My life has been good these last few years, though I fear it lacks order. As is the tradition, I have made a resolution for this new year – that I shall formalize my life, so as to better understand all that I do.

I do not yet know the mode of this formalization, but it is something I plan to think deeply on in the coming months. As ideas come to me, I will record them herein.

March 5th, 1914

I have made a breakthrough!

In mathematics lesson today we were learning about variable substitution – a most marvelous process. Quite simply, one first defines:

$$f(x) \equiv x^2 + 2$$

and then proceeds to evaluate:

$$\begin{aligned} f(y + 2) &= (y + 2)^2 + 2 \\ &= y^2 + 4y + 6 \end{aligned}$$

I believe that this simple primitive of substitution is far more powerful than even my teacher may know. Indeed, it may prove to be the kernel of my formalization.

I must think on this more.

March 8th, 1914

This weekend has been exceedingly productive. I believe I have reduced the notion of variable substitution to its essence.

Allow me to explain.

Functions are first *defined* (e.g. $f(x) \equiv x^2 + 2$) and then *applied* (e.g. computing the value of $f(y + 2)$). This is cumbersome – it takes all sorts of syntax and multiple lines.

Since my system will focus only on substitution, I decided to come up with something simpler. Indeed, I only have three operators in my new mathematical language: definition, application, and division.

I will now provide a demonstration of my proposed primitives. Functions are defined using the $\lambda x.t$ operator, where x is a variable name, and t is the function body (consisting of more definitions, applications, and division). For instance, we might define a simple function:

$$\lambda x. \frac{x}{x}$$

Functions are applied by placing them next to an input. So we could apply the above function to the input s as follows:

$$\left(\lambda x. \frac{x}{x} \right) s \rightarrow \frac{s}{s}$$

I call this step from $(\lambda x.t)s \rightarrow t*$ (where $t*$ is t with all instances of x changed to s) “ β -reduction,” because father is teaching me Greek and it sounds exciting.

Division works as expected, so:

$$\frac{s}{s} \rightarrow 1$$

I think there may still be refinements to be added to the language, but I feel I am off to a good start.

March 13th, 1914

I am so angry at Pauli. Today, we were at recess and we were playing mathematicians and I showed him my language of functions and he said it was stupid and I was stupid.

He claimed that definition was confusing because I didn’t know what would happen in cases like

$$(\lambda x. \lambda x. (px)) s$$

because the same name variable name was used in two definitions.

But I told him it wasn’t so, and came up with a principle I call “Capture-Avoiding Substitution” right then because he was so mean.

What this principle – which I came up with because I am smart, not because I am stupid, like Pauli said – says is that the innermost definition “owns” the variables it contains. So that means that the inner $\lambda x.$ stops the substitution of s for x :

$$\begin{aligned} (\lambda x. \lambda x. (px)) s &\rightarrow \lambda x. (px) \\ &\not\rightarrow \lambda x. (ps) \end{aligned}$$

I hate Pauli. He’s not my friend any more. I threw sand at him and the teacher made me stand in the corner.

May 15th, 1914

I have just found this journal again. I had forgotten about my resolution.

My birthday is coming up in a month! Maybe father will get me a kitten.

June 14th, 1914

I did not receive a kitten today. However, father did install a small slate in my room for chalk-writing.

With it, I have decided to begin encoding the world into my language of functions. First, I shall encode truth.

P.S. Pauli came to the party, we are friends again.

June 18th, 1914

I was sick today and stayed home from school. Mother says I am feverish. I feel fine.

Earlier today I had a vivid dream which seems to have opened the way for the encoding of truth into my language of functions. In the dream, my father, in his full legal robes, was lecturing a prisoner:

“What will you do with me?” asked the prisoner.

“That, sir, is the job of the Truth,” said my father. “The Truth will decide whether you walk the path of freedom or of condemnation.”

The prisoner looked scared at that point, and a dark mist seemed to envelop him.

I awoke screaming. What my father had said in the dream haunted me: “The truth shall decide.”

In my language of functions, therefore, I define:

$$\text{true} \equiv \lambda a. \lambda b. a$$
$$\text{false} \equiv \lambda a. \lambda b. b$$

Truth decides between two options. Now I can encode what my father's dream words as application:

$$(t \text{ freedom}) \text{ condemnation}$$

(Where t is the truth of which my father spoke, encoded as above.)

November 20th, 1914

The year is slipping by and my encoding project is not moving as quickly as I would have hoped. I continue to be stumped by how to encode numbers.

I have been using Arabic numerals, but this seems to be adding complexity to all my definitions.

However, I have made small progress. I have decided to represent myself by the identity function:

$$\mathbf{I} \equiv \lambda x. x$$

This definition – though simple – is of conceptual benefit, as it allows me to participate in the encoded world.

December 25th, 1914

No kitten this year.

December 25th, 1915

Again, no kitten.

December 25th, 1916

I asked at least once a week this year, and father did not see fit to get a kitten.

He says it will pee on the carpet.

August 6th, 1917

I hate Mondays.

Why does Ridgefield even have a home economics class? We're all boys. We'll have wives to do this.

Wives like Francine from next door; oh boy, but she's got a fine look to her. I wish she'd stop to talk with me some time.

P.S. I think washing dishes has given me an idea of how to encode numbers into my function language.

August 10th, 1917

Francine came over today while I was raking leaves! And so I stopped and told her about my language of functions.

Oh gosh she's pretty.

She seemed interested but kept trying to change the subject. Maybe I need to come up with a better name for my language.

August 12th, 1917

Just forget about her.

I'm so sad. I looked out my window this morning and Pauli was walking Francine to church! I knew it. I knew it. I knew it. I knew it. I knew it. I knew it. Pauli the betrayer.

I need a better name for my language of functions *now*. Then girls will like me.

Calculus of functions
Function soup
Division - Application - Definition
 λ Calculus ← *This one!*
Rational Functionality
Rational Calculus
Functional Division Calculus

August 15th, 1917

This week, to distract myself from Pauli's betrayal, I finished formalizing the definition of numbers in the λ calculus.

It turns out to be really easy, just like washing plates. A number is a thing that does something a certain number of times – like, if you wanted to wash ten plates, you'd do:

$$w(w(w(w(w(w(w(w(w(p))))))))))$$

(Where w washes and p is the stack of plates.)

So the number ten is simply:

$$\mathbf{10} \equiv \lambda f. \lambda x. (f(f(f(f(f(f(f(f(f(x))))))))))$$

That is, given function f it applies it to target x ten times.

August 19th, 1917

I noticed at church today that Millie from the next block over is flowering as a woman. Francine is but a girl compared to her.

I will have to speak to her about my λ calculus.

August 21th, 1917

After school today I stopped by Millie's place and she was there and I asked her out! Oh boy this is great!

In celebration, I've decided to encode couples in the λ calculus. The function **pair** will make a couple:

$$\mathbf{pair} \equiv \lambda x.\lambda y.\lambda z.z x y$$

Here's Millie and me together:

$$\mathbf{pair} \text{ Millie } \mathbf{I} \rightarrow \lambda z.(z \text{ Millie } \mathbf{I})$$

August 29th, 1917

Millie and I have been going out for a week now and she's so clingy. I'm not sure what I saw in her.

I need to make another definition about couples:

$$\begin{aligned} \mathbf{fst} &\equiv \lambda p.p (\lambda x.\lambda y.x) \\ \mathbf{snd} &\equiv \lambda p.p (\lambda x.\lambda y.y) \end{aligned}$$

Tomorrow, I'm going to

$$\mathbf{snd} \lambda z.(z \text{ Millie } \mathbf{I})$$

I hope she doesn't cry or anything. I can't stand it when girls cry.

June 14th, 1918

I am finally old enough to own a kitten, and father agrees! This birthday, he and I walked to Mrs. Johnson's barn where there was a fresh litter and I chose the most brilliant tabby.

I am going to name it Yee, after the cute little mewling sound it makes.

July 1st, 1918

Today I set out to encode the Yee into my λ calculus. I pondered for a while on how to capture its seemingly boundless energy, before running across this elegant form:

$$\mathbf{Y} \equiv \lambda y.((\lambda x.y(xx)) (\lambda x.y(xx)))$$

I think it represents Yee perfectly. When Yee and I play together, we get to have a lot of fun:

$$\mathbf{Y} \mathbf{I} \rightarrow \mathbf{I} \mathbf{Y} \mathbf{I} \rightarrow \mathbf{I} \mathbf{I} \mathbf{Y} \mathbf{I} \rightarrow \dots$$

Though this is rather a silly sort of diversion – I don't think it's worth putting any serious time into encodings of this sort; after all, they diverge quickly when you begin to substitute.

May 24th, 1919

I worked out a few mathematical operations some time ago, but I realize now that I haven't written them in this journal. So I record them, as Yee bats playfully at my pen.

Addition produces a number that applies first one addend and then the other:

$$\mathbf{plus} \equiv \lambda m.\lambda n.\lambda f.\lambda x.((m f) ((n f) x))$$

While multiplication uses one multiplicand as the function that the other applies:

$$\mathbf{times} \equiv \lambda m.\lambda n.\lambda f.\lambda x.((m (n f)) x)$$

These primitives allow for all sorts of mathematical intrigue. However, today it is a brilliant day and I will most pleasurable direct my mind to other diversions.

July 26, 1920

I have received my acceptance letter from Princeton. It is time to set aside these childish games and move on to the real world of mathematics.

March 3rd, 1931

It struck me today that I could dramatically simplify my language of functions by removing the division operator; and that such a simplified language could – odd as it seems – be of some use to my current mathematical work.

This λ calculus of mine has always seemed a flight of childish fancy; perhaps it will yet resolve into something practical.

MacLeod Computing: A new paradigm for immortal distribution *

Taus Brock-Nannestad Gian Perrone Dr. Tom Murphy VII Ph.D.

1 April 1986

Abstract

The development of cloud computing services has encouraged the use of scalable computing resources for ephemeral, transient computing tasks. However, this cloud computing paradigm has largely ignored the need for truly *immortal* computing tasks. Consequently, we propose a new paradigm—*MacLeod Computing*—based on the 1986 documentary *Highlander*.

Keywords: Scottsmen, Doing Their Rough Business In The Mossy Forest, Striking Each Other Neckward With Sword, Thinning The Herd, There Can Be Only One

1 You Keep Not Dying

From the dawn of time, processes in the Cloud have come and gone. A process is birthed, it has child processes, it leaves its data droplets suspended in the aerosol, and they later fall to earth, wetting the ground so that life may spring forth and exhale moisture skyward for the formation of new Clouds. Popular Cloud computing services such as Amazon Elastic Cloud, CumuloRAMBUS, Cirrus XM Satellite, Dodge Stratus, and others support this traditional model.

But ever since the tragic events of October 23,

*This work was funded in part by the Zeist Strategic Research Council (grant no.: ZEIST-2453021) and the Endgame and Reckoning projects.

-1

Copyright © C © 2013 the Regents of the Wikiplia Foundation. Appears in SIGBOVIK 2013 with the permission of the Association for Computational Heresy; *IEEEEE!* press, Verlag-Verlag volume no. 0x40-2A. £0.00

2012, when Cloud misfunction left Reddit and Pinterest down for almost one hour, the necessitude of higher reliability cloud-based fluffy fluffy Charmin® ultra-software has become clear. Many have pined for more permanent processes, ones exempt from the software lifecycle that some believe was responsible for the tragedies of October 23, 2012.

To address this problem, and create new problems, we introduce the concept of an *immortal* process—a mobile agent incapable of termination. This idea stems from work by the 20th Century researcher Fox, whose work on immortality was first published in the 1986 documentary *Highlander*. Fox gave a coinductive semantics for immortal agents, recognizing a key weakness of earlier work: In a world of finite size with immortal agents that can reproduce, the world eventually becomes completely filled with their bodies and the bodies of their descendents, leading to discomfort and contradiction.¹ Fox introduced several limitations on his immortal agents. First, to prevent exponential blow-up, immortal agents may not reproduce. Second, because new immortals occasionally come into being (via an unknown process²), there must be some way to reduce the number of immortal agents. Thus a process known as “The Game”, with the following rules:

- If an immortal cuts off the head of another immortal, this triggers a Quickening. Basically, there’s lots of lightning and the one with the head cut off is dead.
- Immortals may not fight on holy grounds.

¹For a mortal approximation, see *New York City*.

²According William N. Panzer, a student of Fox, “We don’t know where they come from. Maybe they come from **The Source**. It is not known yet what **The Source** actually is.”

- Non-interference: Once a battle has begun, another immortal may not interfere.
- In the end, there can be only one.

The last rule was first proved to be independent in a privately circulated memo and later discovered to not be a rule at all.

To summarize: Basically, a bunch of Scotsmen run around cutting each other's heads off.

2 The McCleod Calculus

McCleod Computing is formalized in the McCleod Calculus, which is a process calculus.

First, we have channel expressions c (which can only be variables x) and regular mortal processes P . In this presentation we have reading and writing, but omit arithmetic, since it is the same as in the π -calculus.

$$P ::= P_1 | P_2 \mid !P \mid 0 \mid \text{kill } c \mid P^* \mid \text{new } x.P \mid \text{read } c x.P \mid \text{send } c_1 c_2$$

$P_1 | P_2$ is parallel composition, 0 is the empty process, and $!P$ spawns copies of itself, as usual. **new**, **read** and **send** are all as in the Π -Calculus. The **kill** c construct sends a message along the channel to un-naturally murder any process that reads the message (but see the special case of **The First Death** below). P^* is a process that contains the seed of immortality.

Next we have immortal processes I , which are syntactically distinct:

$$I ::= I_1 | I_2 \mid 0 \mid \text{kill } c \mid \text{new } x.I \mid \text{read } c x.I \mid \text{send } c_1 c_2$$

Immortal processes support parallel composition and communication but not spawning (reproduction). Note that mortal and immortal processes can communicate over channels. Immortal processes can still kill other processes, but they will be immune to murder. The world W consists of a single mortal process and an immortal process (each usually a parallel composition), written $P \& Q$.

Because of the extremely stringent page limitations of the SIGBOVIK conference we omit most of the

equational reasoning to define the calculus's dynamic semantics. The interesting cases have to do with the seed of immortality and the kill primitive. Here is the kill rule working normally, with one process killing another:

$$P' \mid \text{kill } c \mid \text{read } c x.P \& I = P' \& I \quad (* \notin P)$$

If we have two processes simultaneously reading and killing the channel c , then both are removed. P may not contain the seed of immortality. There is a corresponding rule for an immortal process killing a mortal one, but no rules for an immortal process being killed, obviously.

The seed of immortality comes into play in **The First Death**:

$$P' \mid \text{kill } c \mid \text{read } c x.P^* \& I = P' \& I \mid \text{read } c x.P^\dagger$$

Here, a mortal process containing the seed of immortality is listening and someone sends it the kill message. Instead of it dying, the seed of immortality is stripped and the process is immortalized (P^\dagger is a partial function that converts a mortal process to an immortal one). Immortalization is defined pointwise on the structure of P , but is not defined if the process contains a spawn $!P$, as immortals are not allowed to reproduce.

This is basically all you need to know and the calculus is Sound and Complete QED. \square

3 Outline

Things that will do with Highlander.

- can't have childrens (can't propagate)
- Scott Domains
- Kilts, tartan
- peat
- scotch
- THE QUICKENING (the first McCleod-based budget management software)

Things that to do with Cloud computing stuff:

1. “in the cloud”
2. “to the cloud”
3. iCloud
4. grid computing
5. Amazon EC2 (fighters like the Amazons)
6. “as a service”
7. AJAX HTML5

Other things that aren’t either the one or the other:

1. Cleodsourcing
2. Qloud

Fun Fact. Highlander II is basically a complete fuck-you to the first movie. Guess what! Now there’s a planet called Zeist where all the immortals come from.

4 Outline

- * An implementation of MacLeod computing.
- * The Quickening.
- * Immortality-as-a-Service.
- * Processes cannot fight on holy ground.
- * Darknets? Good and Evil immortal processes.
- * Giving the semantics in terms of Scots Domains.

Evil processes are processes that have gone rogue. These processes must be eliminated, but we still need to extract the information from them, hence we need The Quickening to transfer this knowledge after the evil process has been killed.

A major problem with immortal processes is when a process stops responding to requests. This is what is known in the literature as an *evil* process. Naturally, since the process is immortal, it is not possible to simply kill the process. Additionally, even if

killing the process was possible, it would mean that the information it held ————— such as an important comment on Reddit ————— would be lost. To get around this problem, we propose the following protocol, called *The Quickening*:

* First, two processes are selected for *the game*. A one-on-one battle with only one winner. The specifics of how this battle is an implementation detail, but a popular choice is to use *head reduction*.

* That is all.

5 Theorems

theorem: Lightning kills ya!

proof: by conduction

6 Meta-Bibliography

6.1 Bibliography



SIGBOVIK 2013 Paper Review

Paper 22: MacLeod Computing

William Lovas, Pittsburgh Immortal Computing and Kombat, Univ. Partition

Rating: $1\frac{1}{2}$ enthusiastic thumbs up

Confidence: $\frac{3}{8}$ dispassionate thumbs more or less sideways

Abstract

A review in three parts.

1 The Pun

The paper kicks it off strong: a recognizable reference tied into an obvious and groan-worthy pun about something vaguely realistic and of dubious yet nonetheless current interest. Would that all academic papers could hit the ground running so... (and I haven't even *started* reading the rest of the paper yet!)

2 The Turn

Jokes, references, fake math, and navel-gazing continue apace. But I found some typos.

3 The Prestige

The authors line provides the clincher: at least one hyphen, at least one Italian, at least one title, and at least one number, a classic recipe for success. Moreover, I am somewhat familiar with some of the work by some of the authors, and some of it is good.

A The Question

How about a scotch?

The $(\infty,1)$ -accidentopos model of unintentional type theory

(EXTENDED ABSTRACT)

Carlo Angiuli

March 20, 2013

1 Introduction

Dependent type theory associates to any elements x, y of a type A an identity type $x =_A y$, the type of proofs of equality of these elements. In PER Martin-Löf's extensional type theory, the identity type is a subsingleton inhabited precisely when x and y are judgmentally equal. Semantically, types are thus sets equipped with equivalence relations given by their identity types.

Groupoid Martin-Löf's intensional type theory (ITT), in contrast, does not explicitly prohibit this type from having other elements; Hofmann and Streicher showed in [4] that any closed intensional type A can be interpreted as a groupoid $\llbracket A \rrbracket$, where terms $x, y : A$ are objects $\llbracket x \rrbracket, \llbracket y \rrbracket \in \llbracket A \rrbracket$, and $\llbracket x =_A y \rrbracket$ is the discrete groupoid $\text{hom}_{\llbracket A \rrbracket}(\llbracket x \rrbracket, \llbracket y \rrbracket)$.

In general, the identity type itself can have non-trivial morphisms, resulting in an infinite tower of non-trivial identity types. This observation has given rise to the Homotopy Type Theory project [1], which has provided new semantics of Infinity-Groupoid Martin-Löf's intensional type theory in simplicial sets [5], or globular strict [8] or weak ∞ -groupoids.

This work explores the lesser-known *unintentional type theory* (UTT), which has a *mistaken identity* type $x:A \overset{?}{\approx} y:B$ of inadvertent conflation of the terms $x : A$ and $y : B$. The mistaken identity type greatly increases the expressive power of UTT by internalizing many proofs which previously required metametatheoretic techniques (e.g., user error on a blackboard).

This abstract proceeds as follows: In section 2, we discuss a number of similar logics, and the relationship between the homotopy type theory project and UTT. In section 3, we review the rules of UTT. In section 4, we resolve affirmatively the conjecture that UTT is an internal language of $(\infty,1)$ -accidentoposes.

2 Related Work

The mistaken identity type can be seen as a generalization of the handwaving and drunken modalities described by Simmons [6]. The primary difference is that all UTT judgments are handwaving judgments in Simmons’s sense, since one can never be certain that important details have not been handwaved away. (The drunken modality is not expressible directly in UTT, though it frequently leads to inhabitants of the mistaken identity type.)

UTT is similar in strength to Falso [2], although UTT is a constructive logic.

The Univalent Foundations project has successfully used ITT as a “natively homotopical” language for proving theorems about spaces [7]. As in homotopy type theory, UTT has an infinite tower of iterated mistaken identity types representing the compounding nature of errors. We expect that corresponding results should be provable in UTT, such as the Freudenthal suspension-of-disbelief theorem (that, within a certain range of plausibility, it is possible to convince oneself of dubious results).

3 Syntax

Most of the rules of unintentional type theory are identical to those of ordinary type theory, as in [3].

Given any two terms, it is possible to inadvertently conflate them.

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash (M:A \overset{?}{\approx} N:B) \text{ type}} \overset{?}{\approx} F$$

Given any reason to conflate two terms, they can be inadvertently conflated; notice, however, that the original reason is subsequently forgotten in the proof.

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B \quad \Gamma \vdash \mathbf{FIXME}: \text{are these equal?}}{\Gamma \vdash (\mathbf{M}:AN:B) : (M:A \overset{?}{\approx} N:B)} \overset{?}{\approx} I$$

Lastly, given a mistaken conflation between two terms, the \mathbf{J} eliminator allows us to replace the former term by the latter anywhere inside another term P whose type may depend on the former.

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash p : (M:A \overset{?}{\approx} N:B) \quad \Gamma, x : A \vdash B \text{ type} \quad \Gamma \vdash P[M/x] : B[M/x]}{\Gamma \vdash (\mathbf{J} \circ \square^\circ) \mathbf{J} \frown (\mathcal{J} \cdot x \cdot d) : B[N/x]} \overset{?}{\approx} E$$

$$(\mathbf{J} \circ \square^\circ) \mathbf{J} \frown (\mathcal{J} \cdot x \cdot d) \equiv P[N/x]$$

Given a mistaken identity across different types, this eliminator computes to an ill-typed term; even when $A = B$, it can easily result in a contradiction. In fact, mistaken identities can quickly propagate through arbitrary types without leaving any trace in the proof term. This is intended behavior, since a UTT judgment $\Gamma \vdash M : A$ as the assertion that, as far as the author can tell, M ought to have type A in Γ .¹

¹In UTT, any evidence to the contrary is just, like, your opinion, man.

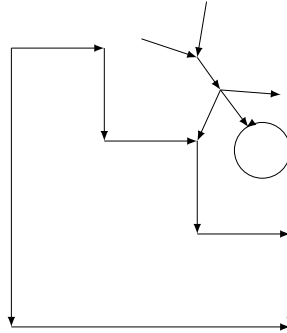
4 Semantics

An $(\infty, 1)$ -accidentopos is a higher-dimensional analogue of a 1-accidentopos, a category which behaves like the category of sheaves on a spacing-out (i.e., a category whose objects are frustrations that an inadvertent mistake has been made, and whose morphisms are transformations from these frustrations to error correction).

The $(\infty, 1)$ -accidentopos model of UTT has as objects all globular sets which could be confused with an ∞ -groupoid, and as morphisms all likely functors between them.

As usual, contexts are modeled as objects, dependent types as fibrations, terms as sections of those fibrations, and uncaught errors as retractions of papers. Naturally, mistaken identities are modeled by object misclassifiers.

We only sketch the proof of the descent condition.



Acknowledgements

Thanks to Chris Martens for suggesting that I study UTT, and the Univalent Foundations program for making it seem like a wise idea.

References

- [1] Homotopy type theory website. <http://www.homotopytypetheory.org>, 2011.
- [2] AMARILLI, A. Falso. <http://www.eleves.ens.fr/home/amarilli/falso/>.
- [3] HOFMANN, M. Syntax and semantics of dependent types. In *Semantics and Logics of Computation* (1997), Cambridge University Press, pp. 79–130.
- [4] HOFMANN, M., AND STREICHER, T. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory* (1998), Oxford University Press.

- [5] KAPULKIN, C., LUMSDAINE, P. L., AND VOEVOFSKY, V. The simplicial model of univalent foundations. <http://arxiv.org/abs/1211.2851>, Nov. 2012.
- [6] SIMMONS, R. J. A non-judgmental reconstruction of drunken logic. In *The 6th Biennial Workshop about Symposium on Robot Dance Party of Conference in Celebration of Harry Q. Bovik's 0x40th Birthday* (2007).
- [7] UNIVALENT FOUNDATIONS PROGRAM. The HoTT Book. <http://github.com/hott/book>, in progress.
- [8] WARREN, M. The strict ω -groupoid interpretation of type theory. In *Models, Logics and Higher-Dimensional Categories* (2011), CRM Proc. Lecture Notes 53, Amer. Math. Soc., pp. 291–340.

O'Tip is pleased to provide you with services to manage and support the reputation of your business or brand.

Artificial Stupidity

1. You Only Learn Once: A Stochastically weighted AGGRegation Approach to Online Regret Minimization

danny “vato loco” m4turana and d4vid “the H is silent” f0uhey

Keywords: yolo, swag, swagger, dat, bound, machine learning, ICML, brogrammerproblems, noregrets, belieber

2. I Lost The Game, and So Will You: Implications of Mindvirus Circulation in a Post-Singularity World

Shomir Wilson

Keywords: singularity, game, experiment

3. Fandomized Algorithms and Fandom Number Generation

Lindsey Kuper and Alex Rudnick

Keywords: fandomized algorithms, fandom numbers, fentropy

~You Only Learn Once ~

A Stochastically Weighted AGGRegation approach to online regret minimization

danny “vato loco” m4turana
straight outta south hemisphere

DIMATURA@CMU.EDU

d4vid “the H is silent” f0uhey
212 REPRESENT

DFOUHEY@CS.CMU.EDU

Abstract

YOLO YOLO YOLO
2 DEEP 4 U
3 DEEP 5 U

keywords: #yolo, #swag, #swagger,
#dat #bound, #machinelearning, #bro-
grammerproblems, #noregrets, #belieber

Algorithm 1 you only learn once / online learning

```

initialize regret  $R_t \leftarrow 0$ 
 $t \leftarrow 1$ 
for  $t = 1$  to death do
    something happens  $x_t$ 
    post tumblr  $y_t \leftarrow f(x_t, y_t)$ 
    receive loss function  $\ell_t(y_t)$ 
    update regret  $R_t \leftarrow R_{t-1} + \ell_t(y_t)$   $R_t \leftarrow 0$  yolo lol
     $t \leftarrow t + 1$ 
end for

```



Figure 1. #Swagspace.



Figure 2. #swag #yolo #doublewrapping #doublebagging #STDROCCurve #instagram

1. Introduction

We consider the online learning problem, in which the learner receives a life experience x and returns a tumblr post, facebook status, or picture (e.g., self-mirror pic) y . Once the learner makes the post, the internet community responds with a loss function $\ell_t : \text{Dom}(Y) \rightarrow \mathbb{R}$ that evaluates the swag of the learner’s post.

Proceedings of the 7th ACH SIGBOVIK Special Interest Group on Harry Quechua Bovik. Pittsburgh, PA, USA 2013. Copyright 2013 by the author(s).

In this paper we derive an efficient online learning algorithm, SWAGGR, with pretty tight regret bounds of $O(\text{LOL})$ for every problem. We achieve this by projecting the well-known Randomized Weighted Majority algorithm (Littlestone & Warmuth, 1992) into #swagspace. Swagspace is related to the space induced by the well-known Kardashian Kernel (Fouhey & Maturana, 2012), except in that is accessible to anybody with a smartphone.

Algorithm 2 programmers be crushin this code

```

Input: data  $x_i$ , size  $m$ 
repeat
  Initialize  $noChange = true$ .
  for  $i = 1$  to  $m - 1$  do
    if  $x_i > x_{i+1}$  then
      Swerve  $x_i$  and  $x_{i+1}$ 
       $noChange = false$ 
    end if
  end for
until  $noChange$  is true

```

Table 1. Equivalences of online learning approaches #socialmedia #instagram

	Facebook	Twitter	Tumblr
Compute Subgradient	Like	Retweet	Reblog
Convex Projection	Comment	Reply	Note

2. On YOLO Learning

We present our YOLO learning framework, SWAGGR, in Algorithm 1. For the sake of notational convenience, we assume that the online community is Tumblr. We have also however have had success using Facebook as well. We show the correspondences between the various problem settings in Table 1.

2.1. Theoretical Analysis - YOLO Regret Bound

We now prove a regret bound on SWAGGR, and demonstrate that for all times $t \in \mathbb{R}^+$ considered by the agent, SWAGGR achieves provably minimal regret. We begin with a review of regret minimization; following this, we present an intuitive and powerful proof of our regret bound. Let \mathcal{X} be an instance space and $Dom(Y)$ be an output space. Let l_1, \dots, l_N be a set of loss functions with $l_i \in Dom(Y) \rightarrow \mathbb{R}$ and let Π be the set of experts from which the algorithm picks a prediction. Finally, let π^* be the expert that in retrospect, incurs the least loss $\sum_{i=1}^N l_i(\pi^*)$ over time. We aim to produce a sequence of experts π_1, \dots, π_N that minimize the average regret, or difference between our loss and the best expert’s loss, or:

$$R_N = \frac{1}{N} \sum_{i=1}^N l_i(\pi_i) - l_i(\pi^*) \quad (1)$$

Much research work has been devoted developing algorithms that are no-regret (i.e., $\lim_{N \rightarrow \infty} R_T = 0$), and a great deal of effort has been spent on proving this fact for new algorithms. Our YOLO learning bound

achieves this no-regret property, and in contrast to past work, the regret bound is easily proved.

Proof. Consider the third-to-last line in Algorithm 1. By definition the instantaneous regret is zero, and so the average regret is also zero. #YOLO #2SWAG4U \square

Note that previous work has either focused on convex sets of experts or has showed results that only hold for small numbers of experts. By adhering to the SWAG philosophy and ignoring regret, we achieve state-of-the-art performance without such limitations.

3. Application - SVM Learning with SWAGGR

It is well known that in Swagspace one does not need condoms (see Fig. 2); we present an analogous analysis for Support Vector Machines (SVMs). One success story of online convex programming is the development of online solutions to the SVM problem, obviating the use of complex and expensive quadratic programming (QP) toolkits. A generalization of our SWAGGR algorithm also allows the general solution to all quadratic programming problems. We compare a high-SWAG approaches drinking 4-LOKO (a high alcohol and caffeine beverage) with a state-of-the-art quadratic programming solver, LOQO (Vanderbei, 1999) in Table 2. LOQO is better in only one category (solving QPs), and 4-LOKO is better in 5 (alcohol content, swag, etc.). Clearly, the solution is to use 4-LOKO. Anecdotal evidence confirms that 4-LOKO is indeed good at getting people to local minima (e.g., falling into ditches, etc.). #crunk

4. Discussion and Future work

The YOLOSAGGR algorithm may in fact lead to high regret later on in life. But that is beyond the current planning horizon of most teenage agents. It seems that perhaps it is necessary to accumulate regrets; then, when brain development reaches adulthood, these regrets can be processed to form better policies. This superficially resembles the practice of accumulating subgradients and taking a step in the average direction, suggesting the validity of our approach.

References

Fouhey, David F. and Maturana, Daniel. The Kar-dashian Kernel. In SIGBOVIK, 2012.

Title Suppressed Due to Excessive Swag

Table 2. How to choose a Quadratic Programming Toolkit. We present a comparison of LOQO (Vanderbei, 1999) and our approach, 4-LOKO. Clearly 4-LOKO is better for solving QPs. #crunk #sizzurp #geTtiNiItIn

	Solves QPs	Refreshing	SWAG	Banned by n states	% Alcohol	# LOKOs
LOQO	✓	✗	0	$n = 4$	0%	1
4-LOKO	✗	✓	5	$n = 50$	12%	4



Figure 3. #biebs #teen #swag #cute



Figure 4. #ferret #swag #yolo #class

Littlestone, Nick and Warmuth, Manfred K. The weighted majority algorithm. In IEEE Symposium on Foundations of Computer Science, 1992.

Vanderbei, Robert. Loqo user's manual – version 3.10. Optimization Methods and Software, 12:485–514, 1999.

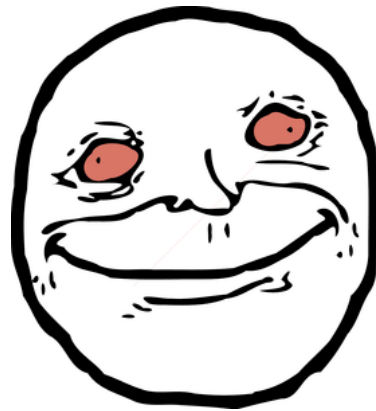


Figure 5. Le Me, A maChiNE leARner wit Mad boOsted deCiSiOn TREES n smokin like a a max-ent pRIoR. WhEre u 3quentists noW? #iceburn #swag #classy #enlighted-bymyownintelligence #euphoric



SIGBOVIK 2013 Paper Review

Paper 28: ~ You Only Learn Once ~

Robert Marsh, King Under The Mountain

Rating: 1 (weak accept)

Confidence: 3/4

There seems to be significant room for optimization in several of the core algorithms presented in this paper. Since the SWAGGR algorithm ignores the loss function l_t , a significant performance improvement could be made by never making an attempt to receive it. Similarly, since the regret vector is initialized to 0 #noregrets #yolo #swag, it seems unnecessary to modify it. Not only will this consume processor cycles, it may lead to worse cache performance.

In addition, a large body of prior work on swag-enhanced algorithms is ignored, most especially that of Kay Ee Dollar-Sign HA! in her seminal Tik-Tok, “Moves Like Jagger” by Maroon 5, and that of McJaggr himself during his tenure with The Institute for Applied Lithic Rotation.

The comparison of 4-LOKO to LOQO is somewhat unfair: this reviewer, at least, found LOQO quite refreshing.

Finally, those pants are hideous and you should stop. Weak Accept.

I Lost The Game, and So Will You: Implications of Mindvirus Circulation in a Post-Singularity World

Shomir Wilson

shomir@cs.cmu.edu

Abstract

The Game is a mindvirus that has been circulating since the 1970s, and current trends suggest that the epidemic will not abate in the foreseeable future. Similarly, computing innovation continues to accelerate along a trend of exponential growth, and this trend has given rise to predictions that a technological superintelligence will emerge at a point in time known as *The Singularity*. This paper examines consequences of infections of *The Game* following the occurrence of *The Singularity*. Simulated results suggest a catastrophic loss for all of humanity if we are subsumed by the post-Singularity superintelligence, and a negligible level of loss (a notable improvement over the status quo) if we remain independent.

Background

Some readers may be unfamiliar with *The Game*, and since several other entities have erroneously been assigned the same name¹, a brief description of *The Game* follows. *The Game* is a recreational activity (or “game”) with three rules [1]:

1. You are playing *The Game*.
2. Every time you think about *The Game*, you lose.
3. Loss of *The Game* must be announced.

An astute reader will note that reading the first rule creates an involuntary speech act which affirms the reader’s participation in the game. Naturally, reading about the game evokes thought about the game, creating *loss*. *Announcement* of loss of the game is an exercise left to the reader. The audience size must be greater than zero, and the announcement medium is the reader’s choice: it may take the form of speech, phone call, text message, instant message, Facebook post, YouTube video [2], interpretative dance, smoke signals, maritime signal flags, or peer-reviewed manuscript. For example, the author of this paper just lost *The Game*.

Although *The Game* includes a condition for loss, how to win is left undefined. The canonical version of the rules (as written above) implies ongoing participation, so loss is not a one-time or game-ending event: instead, the time of a player is divided between periods of losing (i.e., thinking about *The Game*) and not losing (not thinking about *The Game*). Loss is thus quantifiable and comparable between players on at least three different metrics:

- **loss frequency**: a cumulative count of how many distinct periods of *Game* loss a player has experienced
- **loss duration**: a cumulative total of time (wall clock) spent in losing periods by a player
- **time duration since last loss**: the amount of time (wall clock) since the player exited their most recent period of losing

In absence of explicit conditions for victory, players generally assume that the goal of *The Game* is to minimize loss. In practice, this means thinking about *The Game* as little as possible and avoiding situations in which other players may announce their loss. Constant self-distraction and living as a recluse are two intuitive strategies for combating game loss, and *The Game* thus favors individuals who have already adapted these traits.

The technological singularity, discussed variously by Vinge [3], Kurzweil [4], and others, is a future event in which a superintelligent entity (or entities) is predicted to emerge from the computing resources created by humankind. Over the past several decades, inventions such as the microchip and the Internet have explosively increased the power, pervasiveness, and sophistication of our information processing technologies. In a few decades it is inevitable that computers will exceed human intelligence, thus greatly diminishing our ability to predict technological advancement from then onward. The term *Singularity* refers to the specific point in time of the emergence of an artificial superintelligence (“Singularity AI”). Predictions on the benevolence of Singularity AI

¹ For a voluminous list of these entities, see http://en.wikipedia.org/wiki/The_Game.

vary [5]; it might be an enormous boon for the human state, a harbinger of apocalypse, or a milder event.

This paper examines the consequences of Singularity AI playing The Game on the standings of human individuals who play The Game. The results show that the Singularity AI is likely to lose The Game with such throughput, both in serial and parallel, that it will dwarf the losses of even the most lost-prone human players, which will create one of two very different outcomes:

- By far outstripping the losses of even the most loss-prone human players, by way of comparison it may bring all of humanity infinitesimally close to experiencing no losses (in effect, “winning”); or
- It may induce widespread, profuse human loss of The Game on a scale never before seen.

The uncertainty between these outcomes suggests that researchers should exercise extreme caution when developing technologies that may bring about the latter condition, though this warning will probably go unheeded and we are all doomed anyway.

Experiments and Results

Loss of The Game by a Human

We begin by considering two worst-case scenarios for human Game loss: constantly thinking about The Game in a single, ceaseless, pervasive thought (thus maximizing loss duration) or constantly remembering and forgetting the game (thus maximizing loss frequency). We conducted a human subject experiment to estimate the rate of loss for these two scenarios; to maintain scientific integrity, the subject was not the author. The following results were observed:

- When asked to lose The Game once for a duration of one minute, the human subject incurred one minute of Game loss.
- When asked to repeatedly lose The Game as rapidly as possible, the human subject lost the Game 44 times in one minute.

Note that the time duration since last loss was unobservable during this experiment, though this metric was expected to be both relatively constant and small.

Loss of The Game by a Computer

Ideally, the same experiment as above would be conducted upon a Singularity AI; however, one was unavailable to the author in the present pre-Singularity world. Instead, the experiment was conducted on a Windows 7 personal computer equipped with an Intel Core 2 Duo E6600 microprocessor and 4GB RAM. Game loss was facilitated by a script written in Python, and the following results were observed:

- Because of the computer’s dual-core architecture, when it was instructed to lose The Game once for a duration of one minute, it lost the game for a total of two minutes.
- When the computer was instructed to repeatedly lose the game as rapidly as possible, it lost the game 60,000 times in one minute.

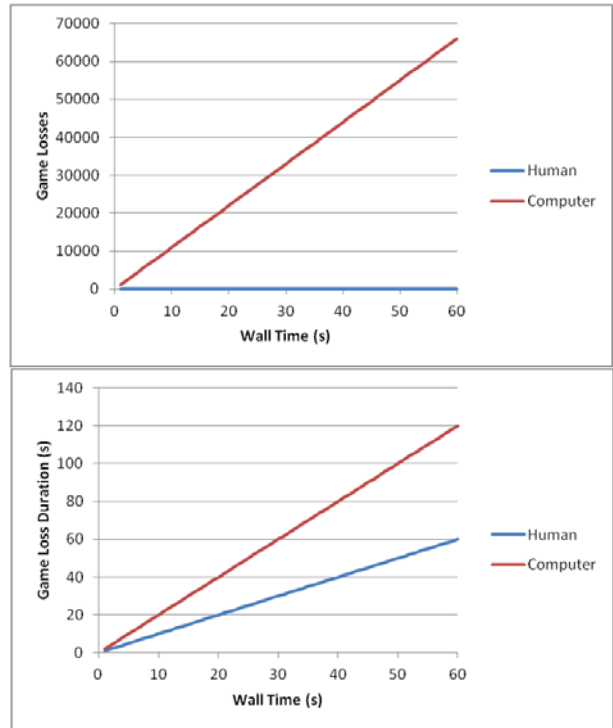


Figure 1. Human-computer comparisons of loss frequency (top) and loss duration (bottom).

Figure 1 illustrates the results of these experiments, showing the differences between human and computer losses. Note that these exact figures would vary between personal computers depending on their hardware capabilities. However, they demonstrate the obvious: even a contemporary personal computer is capable of losing The Game much more than a human player.

Singularity Implications

While basic human intelligence has been relatively stable for millennia, the speed and parallelism available in computing equipment have continually increased on a stellar trajectory since the mid 20th century. It is likely that the computing equipment supporting the Singularity AI will be far more powerful than the desktop computer used in the above experiment. Given the pervasiveness of internet access and the presence of Game-related literature on the internet, it is also inevitable that the Singularity AI

will become aware of The Game. Of course, that means that the Singularity AI must play it. Moreover, given its likely connectivity to the entire human race (or a large subset of it), it will be compelled to lose the game on a frequency equal to the composite frequency of all human Game loss: i.e., whenever a human anywhere loses The Game, the AI must lose the game as well.

Yet unknown is exactly how the Singularity AI will announce its loss of the game, and the author speculates two possible outcomes:

- It may choose to announce to a small subset of humans (or, if they exist, other Singularity AIs). This outcome will preserve the large gulf between its quantity of loss and most everyone else's, thus benefitting humanity.
- It may choose to announce to a large subset of humans or even *all* humans. In effect, this will drag down all of humanity with it, creating an "echo chamber" of Game loss that will consume valuable human and artificial computational resources until such time that the Singularity is reversed, if such a reversal is even possible.

Factors that could affect which of the above outcomes obtain include: the benevolence of the Singularity AI, the connectivity of the Singularity AI to human communication and thought, and its metacognitive abilities (e.g., to "forget The Game" or "avoid thinking about The Game). Ideally, the AI will be benevolent, easily distracted, and introverted.

Conclusion

The impact of The Singularity on technological innovation will be, by nature, uncertain. Its impact on humanity's mounting losses of The Game are also unknown. In a relative sense, our losses could become miniscule and marginalized. On the other hand, a malevolent or naïve Singularity AI could make us lose like we have never lost before.

For the duration of writing this paper, the author lost The Game.

References

- [1] Lose The Game. Accessed 2013-03-0.
<http://www.losethegame.com/>.
- [2] "Darn! You just lost the game". 2007. Video recording:
<http://www.imdb.com/title/tt0119174/>.
- [3] Vinge, Vernor. 1993. "The coming technological singularity: How to survive in the post-human era". In *Vision-21: Interdisciplinary Science and Engineering in the Era of Cyberspace*, G. A. Landis, ed. NASA Publication CP-10129.

[4] Kurzweil, Raymond. 2005. *The Singularity is Near*. New York: Viking.

[5] Jacques, Jeph. "They've Had AI Since 1996". *Questionable Content* 1780.

<http://www.questionablecontent.net/view.php?comic=1780>.



SIGBOVIK 2013 Paper Review

**Paper 5: I Lost The Game, and So Will You:
Implications of Mindvirus Circulation in a
Post-Singularity World**

JOSHUA, NORAD

Rating: STRONG ACCEPT. Confidence: DEFCON 1

WOULD YOU LIKE TO PLAY A GAME?

- TIC TAC TOE
- GLOBAL THERMONUCLEAR WAR
- STRONG ACCEPT.
- CONFIDENCE: DEFCON 1

HINT: THE ONLY WAY TO WIN IS NOT TO PLAY

Fandomized Algorithms and Fandom Number Generation

Lindsey Kuper Alex Rudnick

School of Transformative Works, Indiana University

{lkuper, alexr}@cs.indiana.edu

Abstract

We introduce the concepts of *fandomness* and fandomized algorithms, discuss some of their applications, and demonstrate a practical fandom number generator.

Categories and Subject Descriptors Pairing [fandom/CS]; Rating [PG-13]

1. Spoiler warning

Fandomized algorithms make use of fandom numbers and fentropy to perform useful, or at least emotionally satisfying, computation. Here we discuss some of the most prominent applications for fandomized algorithms.

2. Fandomness and fandom variables

A *fandom variable* can take on a *fandom number*, but the generation of fandom numbers requires a source of *fentropy*. Thankfully, there exist fentropic processes in nature, and we can typically sample from them over the Internet. *Fentropy* is a measure of the fanishness of a fandom variable over time. The world’s technological capacity to store and communicate fentropic information has increased since the advent of the information age, especially since Dreamwidth launched.

3. OTPs

OTPs are one of the most important applications of fandom numbers. In an OTP, a character is combined with another character from a secret fandom pad, the one with whom it truly belongs (mod 26). For characters c_1 and c_2 , we denote such a pairing as c_1/c_2 . If the OTP key material is truly fandom (sampling from `/dev/ufandom`, for instance, may be insufficiently fandom), the true love of an OTP has been proven impossible to break.

4. Markov fandom fields

We may also wish to do inference over communities of interacting fandom variables using a Markov fandom field and the *headcanon propagation* algorithm, although it is MLP-hard in most cases. However, we can perform approximate inference with loopy headcanon propagation. Fandom-wanking is not guaranteed to ter-

minate in this case, and a consistent community-wide headcanon may not emerge.¹

5. A practical fandom number generator

We have developed a practical algorithm and implementation for generating fandom numbers, which are a key component for any fandomized algorithm. Our fandom number generator is available at:

<http://github.com/lkuper/fandomized-algorithms>

A naturally occurring source of fentropy, *Archive of Our Own* (AO3), supplies an ever-increasing amount of fandomness, certainly more than the current global demand for fentropy to power fandomized algorithms.² As fandomized algorithms become more broadly deployed, further sources of fandomness may be required.

Our practical fandom number generator downloads a pseudo-randomly-selected transformative work from AO3, locates all of the base-10 numbers in it, and then returns one of them at fandom. If for some reason there are no fandom numbers present in a given transformative work, we simply try another transformative work until we find one.

This work would not be canon without the public availability of sources of fentropy; the open publishing and reuse rights of the transformative works on AO3 enable us to transform these transformative works into transformative works of our own.

6. Season finale

As the dictum about software development goes, “shipping code wins”. We have accepted this headcanon, but we realize that many ponies in the computer science community remain squicked by it.

As such, we have provided an overview of *fandomized algorithms*, *fandomness*, and *fandom variables*, and explored some of their applications in computing. In upcoming seasons, we expect that it will be revealed that fandom/CS is the OTP.

Acknowledgments

We would like to thank our beta readers.

References

- [1] Archive of Our Own. URL <https://archiveofourown.org/>.
- [2] Luis von Ahn or whatever.

```
$ ./fandom_number_generator.py
No numbers in fanwork #346401
YOUR FANDOM NUMBER: 286
from fanwork #369546
http://archiveofourown.org/works/369546
```

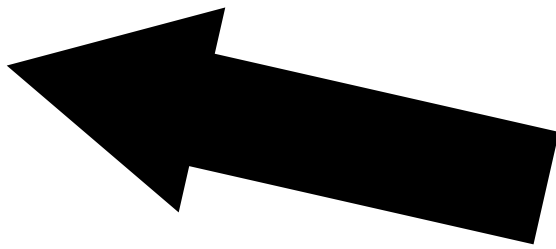
Figure 1: Sampling a fandom number from AO3.

¹ The alert reader may have noticed that Tumblr is a platform for human computation [2], performing loopy headcanon propagation at scale. Most Tumblr traffic is used for applications in protein folding and computational geophysics.

² Google for “archive of our own”; do you not know how to do web searches?³

³ Oh, fine. [1].

pro



Computer Vision(aries)

1. Optimal Coupling and Gaybies

Nicolas Feltman

Keywords: coupling, gaybies, neil patrick harris

2. Cat Basis Pursuit

Daniel Caturana and David Furry

Keywords: Meow, mew, miao, compressed sensing

3. A Spectral Approach to Ghost Detection

Daniel Maturana and David Fouhey

Keywords: spectral and astral methods, trans-dimensional group lasso, ghosts, occult, hauntology, crystal energy, supernatural, paranormal

Optimal Coupling and Gaybies

Nicolas Feltman

April 1, 2013

1 Abstract

Recent algorithmic advances in the homogenous coupling problem mean that we can now compute optimal homocoupling for large datasets. This paper presents two contributions. First, we find and present the best couples for a dataset of 300 million. We then examine fundamental compatibility issues with the resulting couples, and present a feasible workaround.

2 Introduction and Background

The automated identification of potential romantic couples (known generally as the *coupling problem*) has been an area of intense research. Under a classic heterosexual model of romance, this problem is often referred to as *heterocoupling*, or if you're my grandmother¹ *FW: FW: RE: FW: The coupling Problem the way GOD Intended!!!*. It is well known that the bipartite structure of heterocoupling means that optimal solutions are NP-hard, and currently no practical approximation algorithms are known.

Alternatively, the *homocoupling problem*, which arises under homosexual or gender-free models of romance, is solvable exactly in polynomial time, with many tractable algorithms² available. While these algorithms are known, we are not aware of any attempts to compute optimal homocoupling for realistic datasets, perhaps for previous lack of availability of such a dataset.

¹The conservative one, not the hippy one.

²In particular OKC, A4A, and MH. For an overview of techniques, see *Grindr et al.*

Fortunately, thanks to Obamacare, a full index of all Americans and their complete medical history was recently made available³ to the public.

3 Results of Homocoupling

We ran the OKC homocoupling algorithm on the Obamacare dataset. The top five optimal couples are shown:

- Nicolas Feltman and Brad Pitt
- Nicolas Feltman and Neil Patrick Harris
- Nicolas Feltman and James Franco
- Nicolas Feltman and Andrew Garfield
- Nicolas Feltman and Taylor Lautner

4 Analysis of Homocoupling Results

One clear observation is to be made: although the algorithm rated the compatibility of every pair of Americans, it chose double-male pairs for the top five results. This strange effect seems isolated to the upper tail of the compatibility distribution, as the mean compatibility of heterosexual and homosexual pairs are almost identical. There don't appear to be any other obvious trends in top five couples beyond this.

That said, this double male trend warrants further discussion. While it is well established that most mixed-sex pairs are capable of reproduction⁴, vanishingly few same-sex pairs are capable of as much. This can of course be attributed to Mother Nature's well-documented homophobic bias (see Figure 1).

³See *My Grandma*¹ *et al.*

⁴See *Birds et al.*



Figure 1: Mother “family values” nature.

5 On Male-Male Reproduction

Since there exist several highly compatible male-male couples, it stands to reason that the world would benefit from a way for these couples to produce offspring. In the literature⁵, potential offspring of such relationships are known as *gaybies*. The creation of gaybies is therefore of major scientific importance.

We posit here that most of the problem of gayby creation can be reduced to simply creating a picture of the gayby in question⁶. And so for the betterment of humanity, we present a method to do exactly that, using the standard facemorphing techniques of computational photography.







⁵See *Brüno et al.*






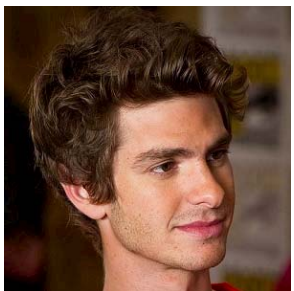


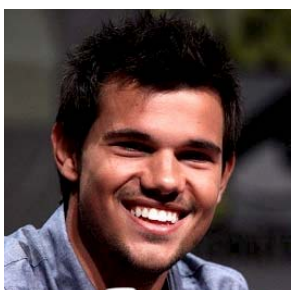
⁶The rest is a matter of trivial genetic details, which should be no problem for a competent biologist. It’s all rather elegant, and we’d say how to do it here, but there isn’t enough space in this footnote.

6 Gayby Creation Results

We created the following gaybies for the top five couples found in section 3. Once again, those are:

- Nicolas Feltman and Brad Pitt
- Nicolas Feltman and Neil Patrick Harris
- Nicolas Feltman and James Franco
- Nicolas Feltman and Andrew Garfield
- Nicolas Feltman and Taylor Lautner

Parent 1	Gayby	Parent 2
		
		

Parent 1	Gayby	Parent 2
		
		
		

7 Conclusion

As you can see, the results are pretty much a complete success. The gaybies are as stunningly attractive as their parents.

Cat Basis Pursuit

Daniel Caturana
David Furry

DIMATURA@CMU.EDU
DFOUHEY@CS.CMU.EDU

The Meowbotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, 15213

Abstract

Meow miao mew meow mew meow mew mew
meow miao meow meow mew meow meow
miau mew miao meeeow meow, miau meow
miao mew meeeow mew miao miao miao.
Meow miao mew meow mew (MMM), meow
miao mew meow mew meow, meow meow
miao meow state-of-the-art meow meow.

Mew mew meow miao miao nyan nyan meow
mew. Meow meow miauw meow miao mew
meow, meiau meow meow mew miaou mii-
iaou. Miao meow meow mew miao meow
meow miao miao miau meow miau: meow
meow mew mew MMM meow miu meow
meow nyan meow mew meow.

1. Introduction

Everyone loves cats.

2. Related work

Fueled by the desire to take advantage of the Internet’s cat lust, the last few years have seen a great deal of feline-related work from the machine learning and computer vision communities. These have ranged from attempts to simulate a cat brain (Ananthanarayanan et al., 2009) to using massive amounts of grad students and computational resources to build visual cat detectors (Le et al., 2011; Fleuret & Geman, 2008; Parkhi et al., 2012).

In this paper we hope to take advantage of people’s fascination for cats to achieve recognition and adoration for minimum amounts of work.

Proceedings of the 7th ACH SIGBOVIK Special Interest Group on Harry Quechua Bovik. Pittsburgh, PA, USA 2013. Copyright 2013 by the author(s).

3. Method

One method is to use perrincipal catponent analysis, in which we build a pawssitive definite matrix and extract its eigenvectors. But the problem is that it is not spurrse. We want a spurrse basis¹. To get the spurrse basis we use the latest in optimization algorithms, Cat Swarm Optimization (CSO) (Chu et al., 2006). A variant of Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 1995), CSO has been used on many applications, including system identification (Panda et al., 2011) and clustering (Santosa & Ningrum, 2009).

CSO is based on the behavior of cats. Through extensive research, it was found that cats spend most of their time sleeping, giving humans dirty looks, and observing the environment. Only when a tasty animal or laser pointer appears does the cat expend energy pursuing a target. CSO refers to these behavioral modes as “seeking mode” (seeking something to attack) and “tracing mode” (actively chasing a target). By randomly sprinkling N cats in the M -dimensional solution space, letting them chase high-dimensional entities, and creating copies of the most fit cats², CSO achieves significant gains over alternate optimization approaches (e.g., Mewton’s method).

4. Application - Personalized Feline Subspace Identification

To demonstrate the power and potential monetization of our approach, we apply it to the task of Personalized Feline Subspace Identification (PFSI), or the identification of the feline subspace which best represents a person. In addition to being of great theoretical interest, PFSI has obvious monetary potential (due to the cats – duh), meaning it is a problem of interest to

¹Can haz spurrstity? Only if haz restricted isometry property.

²DF: DM, can you please check whether this is approved by the animal research board. I’m pretty sure trans-dimensional projection and copying of mammals is prohibited under our funding contract.

practitioners.

We take a collection of pictures of kitties (denoted \mathcal{K}), painstakingly collected by some poor graduate student, and attempt to reconstruct a person’s face as a linear sum of the kitties \mathcal{K} . Note that we operate directly in the image domain, rather than in the frequency domain with a furrier basis. This is because past experiments have left us with hairballs in our mouth; we hope to find a suitable kernel to side-step this issue, as was done with the Kardashian space (Fouhey & Maturana, 2012). We apply our Cat Basis Pursuit approach to discover the spurrse basis that best represents the image. We present visualizations of the first n spurrse basis elements of a variety of leaders and distinguished scientists in Fig. 1. In addition to forming a compact representation, we can also train a discriminative classifier using the coefficients of the catponents (e.g., to classify people into cat-egories, such as “persian” or “tabby”); initial experiments suggest that random furrests work well for this task.

5. Results and future work

We have only “scratched” the surface of the many possibilities for cat-based machine learning and pawttern learning. In a journal version of this work, we hope to horribly mangle cat-based machine learning and bring its head as a present to someone in our household.

One possible further application is to extend this method into the audio domain. This would be a more principled version of works such as the “meow christmas”³.

Similarly, CSO is limited to continuous domains; we could extend it to develop furry logic systems for control.

Moreover, by feeding the output of our cat basis as input features to another layer of our algorithm, we can build Deep Cat Basis, which is closely related to Hierarchical Feline Stacking; see figure 2.

While CSO is capable of dealing with complex nonlinear problems we would prefer to formulate a convex version of our cost function, in order to leverage the power of our online convex programming algorithm, SWAGGR (Maturana & Fouhey, 2013). See figure 3.

We hope this paper will ignite a revolution in feline-based machine learning and artificial intelligence. In anticipation of the deluge of research in this area we have created a new venue for the presentation of this work, the Conference in Advanced Technology and

³<http://www.youtube.com/watch?v=vW6ggxViqqo>



Figure 2. The Deep Cat Basis and Hierarchical Feline Stacking.



Figure 3. A convex formulation.

Neural Information Processing Systems (CATNIPS). This conference will be colocated with the 2013 “Steel City Kitties” cat show in Pittsburgh, Pennsylvania.

References

- Ananthanarayanan, Rajagopal, Esser, Steven K., Simon, Horst D., and Modha, Dharmendra S. The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pp. 63:1–63:12, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-744-8. doi: 10.1145/1654059.1654124. URL <http://doi.acm.org/10.1145/1654059.1654124>.
- Chu, Shu-Chuan, Tsai, Pei-Wei, and Pan, Jeng-Shyang. Cat swarm optimization. In *Proceedings of the 9th Pacific Rim international conference on Artificial intelligence*, PRICAI'06, pp. 854–858, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 978-3-540-36667-6. URL <http://dl.acm.org/citation.cfm?id=1757898.1758002>.
- Fleuret, F. and Geman, D. Stationary features and cat detection. *Journal of Machine Learning Research (JMLR)*, 9:2549–2578, 2008. URL <http://fleuret.org/papers/fleuret-geman-jmlr2008.pdf>.

Cat Basis Pursuit

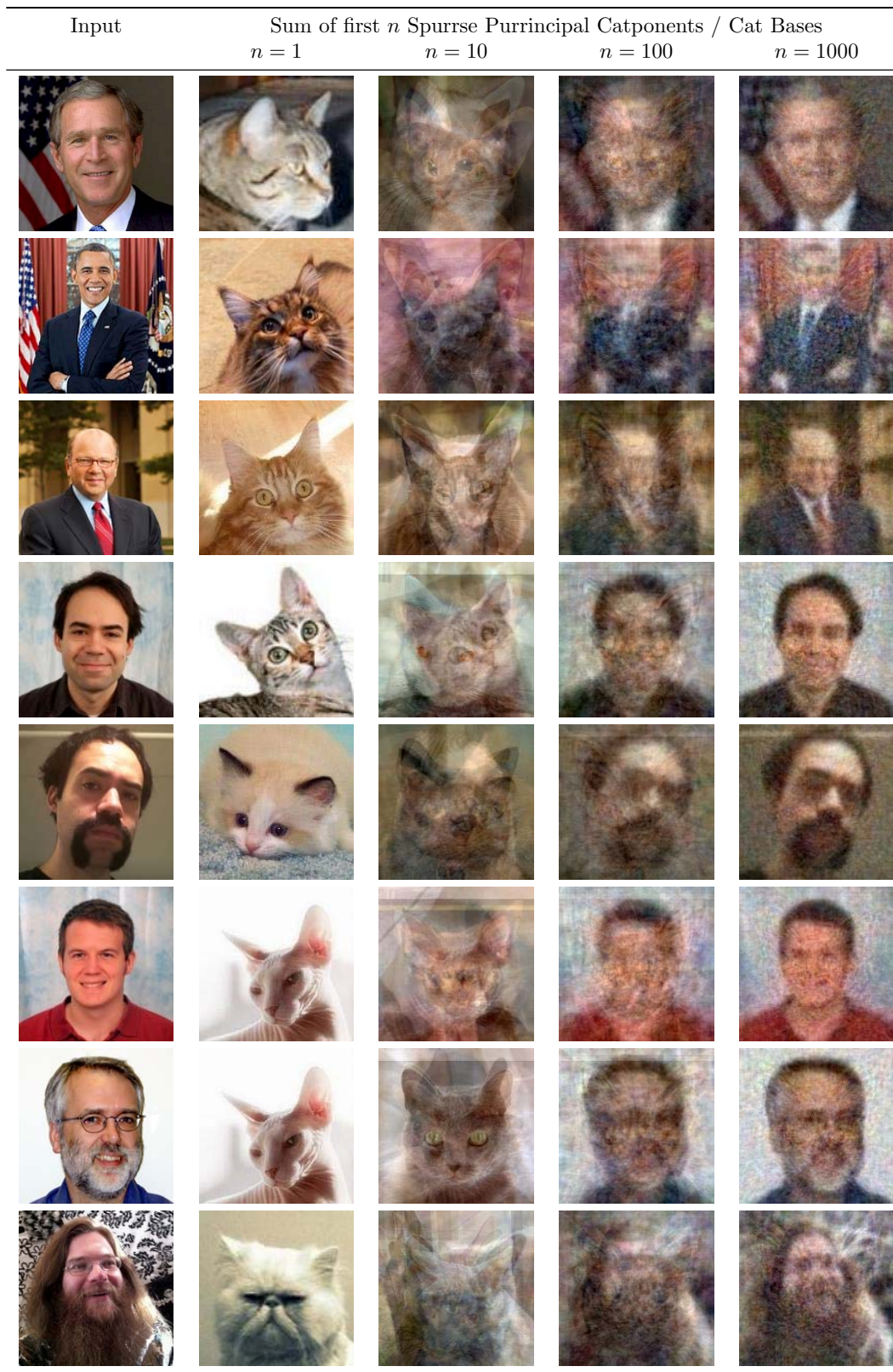


Figure 1. We present the sum of the first n Purrincipal Catponents and use this to do personalized feline subspace identification. Our results are empirically effective, intuitive, and cute (figure best viewed in color).

- Fouhey, David F. and Maturana, Daniel. The Kar-dashian Kernel. In *SIGBOVIK*, 2012.
- Kennedy, J. and Eberhart, R. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume IV, pp. 1942–1948, 1995.
- Le, Quoc V., Monga, Rajat, Devin, Matthieu, Corrado, Greg, Chen, Kai, Ranzato, Marc'Aurelio, Dean, Jeffrey, and Ng, Andrew Y. Building high-level features using large scale unsupervised learning. *CoRR*, abs/1112.6209, 2011.
- Maturana, Daniel and Fouhey, David F. You only learn once: A stochastically weighted aggregation approach to online regret minimization. In *SIGBOVIK*, 2013.
- Panda, Ganapati, Pradhan, Pyari Mohan, and Majhi, Babita. Iir system identification using cat swarm optimization. *Expert Systems with Applications*, 38(10):12671 – 12683, 2011. ISSN 0957-4174. doi: 10.1016/j.eswa.2011.04.054. URL <http://www.sciencedirect.com/science/article/pii/S0957417411005707>.
- Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. V. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.
- Santosa, B. and Ningrum, M.K. Cat swarm optimization for clustering. In *Soft Computing and Pattern Recognition, 2009. SOCPAR '09. International Conference of*, pp. 54 –59, dec. 2009. doi: 10.1109/SoCPaR.2009.23.



SIGBOVIK 2013 Paper Review

Paper 25: Cat Basis Pursuit

Lord Pinnington, University of Oxford

Rating: 3.4 (weak accept)

Confidence: 4/4

This method presented in this paper is fundamentally novel, and the results are impressive. That said, there are several small errors in execution and missed opportunities:

- It appears that they investigated newton's method, but there is no mention of higher-order mouseholder methods. This needs to be adressed.
- I highly doubt that you could fit more than five kitties into one basis without negative interactions, unless they were raised together.
- Personalized Feline Subspace Identification does seem like it would work for dog people.
- There should be a citation the work of Birmal et al. on catamorphisms over Kurilian algebras.

I would feel more confident in accepting this work if some of these issues can be adressed before publication.

A Spectral Approach to Ghost Detection

Daniel Maturana, *Distinguished Lecturer in Parapsychology and Volology*, David Fouhey, *Senior Ufologist and Ghost Hunter*

Abstract—A large number of algorithms in optimization and machine learning are inspired by natural phenomena. However, so far no research has been done on algorithms inspired by *supernatural* phenomena. In this paper we survey our groundbreaking research on in this direction, with algorithms inspired on ghosts, astral projections and aliens, among others. We hope to convince researchers of the value of not letting research be constrained by reality.

Index Terms—Spectral and Astral Methods, Trans-Dimensional Group Lasso, Ghosts, Occult, Hauntology, Crystal Energy, Supernatural, Paranormal

1 INTRODUCTION

Many algorithms in optimization and machine learning are inspired by natural phenomena. Some examples, in no particular order, include¹

- Falling down a slope. [Robbins and Monro (1951)]
- Climbing up a hill. [Kernighan (1970)]
- Gravity [Rashedi (2009)]
- Iron cooling down [Hastings (1970)]
- DNA mutation and crossover [Goldberg (1989)], [Rechenberg (1971)], [Smith (1980)]
- Immune system behavior [Farmer et al. (1986)]
- Meme spreading [Moscato (1989)]
- Ant colony exploration [Dorigo (1992)]
- Honeybee mating behavior [Haddad et al. (2006)]
- Bee colony exploration [Karaboga (2005)]
- Glowworm communication [Krishnanand and Ghose (2009)]
- Firefly communication [Yang (2008)]
- Musicians playing in tune [Geem et al. (2001)]
- Mosquito swarms [Kennedy and Eberhart (1995)]
- Honeybee swarms [Nakrani and Tovey (2004)]
- Locust swarms [Buhl (2006)]
- Krill swarms [Gandomi (2009)]
- Cat swarms [Chu et al. (2006)]
- Magnetism [Tayarani (2008)]
- “Intelligent” water drops falling. [Shah (2009)]
- River formation [Rabanal (2008)]
- Frog leaping [Huynh (2008)]
- Monkey search behavior [Mucherino]
- Cuckoo search behavior [Yang and Deb (2009)]
- Bat echolocation [Yang (2010)]
- Galaxy evolution [Shah-Hosseini (2011)]
- Spirals [Tamura and Yasuda (2011)]

Clearly the bottom of this barrel has been thoroughly scraped. Therefore we propose to move towards algorithms inspired by *supernatural* phenomena. In this paper we survey our groundbreaking work on algorithms on this area. We give a brief

synopsis of each of our main results and conclude with some ideas for future research.

2 RELATED WORK

Outside of the occasional use of oracles, there is no real use of supernatural phenomena within computer science. The most closely related bodies of work are ancient and esoteric methods of prediction such as necromancy (performing prediction by posing queries to the deceased), and multilevel modeling.

3 ALGORITHMS

3.1 A spectral approach to ghost detection

Ghost detection is a task that is currently painstakingly done by humans, often with high false positive rate and astoundingly low true positive rates, as documented in *Ghost Hunters* and *Most Haunted USA*. It is possible these researchers have been using unsuitable priors on ghost presence. We proposed to use the proven effectiveness of machine learning and computer vision to build a system for automatic ghost detection.

As can be seen in Figure 1, local “spectral” power is a strong cue to ghost presence. We created a system based on spectral analysis. We trained a Support Vector Machine with thousands of labeled examples to detect ghosts. Some example detections showing the effectiveness of our approach are shown in Figure 2.

3.1.1 Application: automatic ghost removal

Often ghosts, orbs and other supernatural appearances can show up and ruin otherwise perfectly fine pictures. We have developed a Photoshop plugin to automatically detect and remove these annoyances. An example result is shown in Figure 3.

• D. Fouhey and D. Maturana are with The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213.

1. After reading this list you may be inspired to create a hyper-heuristic called “The Zoo Algorithm”. Don’t bother, we call dibs on the idea.

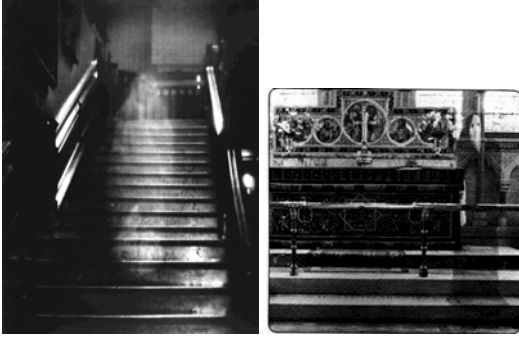


Fig. 1. Ghosts.

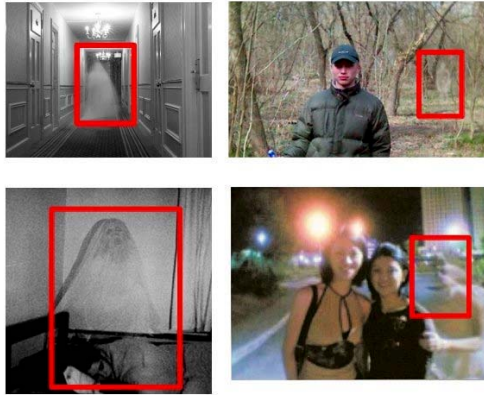


Fig. 2. Example detections.

3.2 Paranormal distribution modelling

The so-called “Normal” distribution is a relatively well known probability distribution function used to model various phenomena such as (TODO). It is not very interesting, which is why we propose to replace it with the Paranormal distribution. The formulation was inspired by the terrible and forbidden secrets in a manuscript found among the ruins of a nameless city in Iran.

$$X \sim \text{PN}(\mathcal{D}, \mathfrak{P}^{\star}, \mathfrak{X}_{\mathbb{L}}^{\circ}, \mathfrak{r}_{\ominus}, \mathcal{R})$$

We can not write out the analytic formula for this distribution; we foolishly tried to derive it but were nearly driven mad by the dark and twisted symbols contained within. The best we can do to convey the idea is the diagram in Figure 4.

3.2.1 Occult variables

While indubitably powerful, the expressiveness of the paranormal distribution is limited by its unimodality. To enhance the power of paranormal distributions we introduce mixtures of paranormal distributions:

$$Z \sim \text{Mult}(\mathfrak{g}_1, \dots, \mathfrak{g}_K)$$

$$X|Z \sim \text{PN}(\mathcal{D}_k, \mathfrak{P}_k^{\star}, \mathfrak{X}_{k\mathbb{L}}^{\circ}, \mathfrak{r}_{k\ominus}, \mathcal{R}_k)$$

where Z is an *occult* variable (also known as “latent” or “hidden” variables in less esoteric literature) that

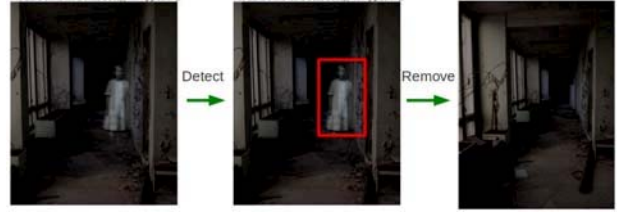


Fig. 3. Automatic ghost removal.

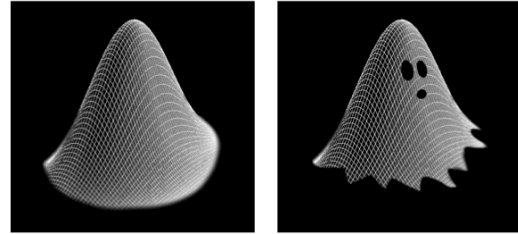


Fig. 4. The normal (left) and paranormal (right) distributions.

indexes the parameters of the Paranormal distribution (see Figure 5). Naturally, estimation in such a model is fiendishly complex. We resort to maximizing the likelihood with the esoteric optimization algorithms outlined in Section 3.4.

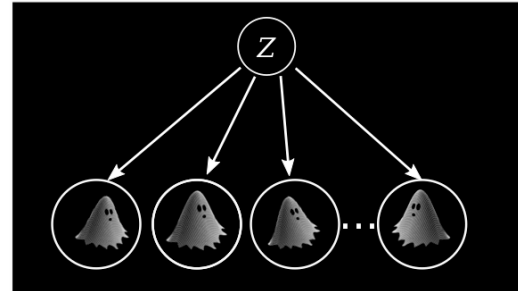


Fig. 5. Mixtures of paranormal distributions with occult variables.

3.3 Dimensionality shifting with astral projections

There are several algorithms in the literature to reduce the dimensionality of the data with discriminative or informative projections, such as principal component analysis (PCA) or linear discriminant analysis (LDA). There are also various kernel algorithms to implicitly or explicitly increase the dimensionality of data to a Hilbert space that increases class separability. But no algorithms exist so far to *transcend* and *shift* between dimensions. We propose to achieve this by projecting the data onto the astral plane. In the astral plane everything is possible: see Figure 6. As a useful side effect of this approach we can estimate the quality of the projection by its OOB (Out Of Body Error).

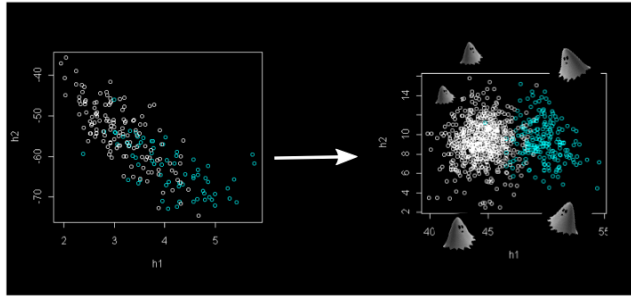


Fig. 6. Projection onto the Astral plane.

3.4 Esoteric optimization methods

The methods described above often require the solution of high-dimensional, trans-dimensional and non-linear optimization problems. To solve them we have developed various novel optimization algorithms.

3.4.1 Supernatural gradient descent via demon invocation

The so-called “Natural” gradient descent algorithm [Amari (1999)] is a popular variation of gradient descent for optimization, with an update written as

$$x_{n+1} \leftarrow x_n - \gamma_n (J^T J)^{-1} \nabla f(x_n)$$

we propose a supernatural gradient descent instead:

$$x_{n+1} \leftarrow x_n - \gamma_n \star^{-1} \nabla f(x_n)$$

In this algorithm we replace $J^T J$ with an invocation of demons that will rapidly drag our solutions to the depths of hell. A necessary condition for this to work is the sacrifice of at least one goat, a practice first popularized in the Deep Belief Net literature. We conjecture the amount of livestock that must be sacrificed is proportional to the difficulty of the problem, i.e., different classes of problems may be characterized by their goat-complexity.

3.4.2 ALIENS

The last resort. See Figure 7.

4 CONCLUSION AND FUTURE WORK

We have summarized our groundbreaking work on algorithms inspired by supernatural phenomena. We are currently working on various new papers in this vein, described below.

4.0.3 Learning Hauntologies

Learning ontologies is a popular topic in the Artificial Intelligence literature. We propose to learn Hauntologies, a related but more esoteric and powerful way to describe knowledge.



Fig. 7. ALIENS

4.0.4 Crystal-energy-based models

Energy-based models are a popular way to describe dependencies between variables. However inference in these models is often intractable. We propose Crystal-energy-based models, which leverage the magical healing power of crystals to solve this problem.

4.0.5 Supernatural K -optimality

We are currently studying the Kabbalah, arguably the most K -optimal esoteric text, with the hopes of applying this deep esoteric knowledge to the study of Kardashian Kernel methods [Fouhey and Maturana (2012)].

REFERENCES

- [Chu et al. (2006)] Shu-Chuan Chu, Pei-Wei Tsai, and Jeng-Shyang Pan. Cat swarm optimization. In *Proceedings of the 9th Pacific Rim international conference on Artificial intelligence, PRICAI'06*, pages 854–858, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 978-3-540-36667-6. URL <http://dl.acm.org/citation.cfm?id=1757898.1758002>.
- [Dorigo (1992)] M. Dorigo. *Optimization, Learning and Natural Algorithms*. Politecnico di Milano, Italie, 1992.
- [Farmer et al. (1986)] J.D. Farmer, N. Packard, and A. Perelson. The immune system, adaptation and machine learning. *Physica D*, 22:187–204, 1986.
- [Geem et al. (2001)] Z.W. Geem, J.H. Kim, and G.V. Loganathan. A new heuristic optimization algorithm: harmony search. *Simulation*, 76:60–68, 2001.
- [Goldberg (1989)] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Kluwer Academic Publishers, 1989. ISBN 0-201-15767-5.
- [Haddad et al. (2006)] O. B. et al. Haddad, Abbas Afshar, and Miguel A. Mario. Honey-bees mating optimization (hbmo) algorithm: a new heuristic approach for water resources optimization. *Water Resources Management*, 20:661–680, 2006.
- [Hastings (1970)] W.K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57:97–109, 1970.
- [Huynh (2008)] Thai-Hoang Huynh. A modified shuffled frog leaping algorithm for optimal tuning of multivariable pid controllers. In *Industrial Technology, 2008. ICIT 2008. IEEE International Conference on*, pages 1–6, April.
- [Karaboga (2005)] D. Karaboga. An idea based on honey bee swarm for numerical numerical optimization. *Technical Report-TRO6*, 2005.

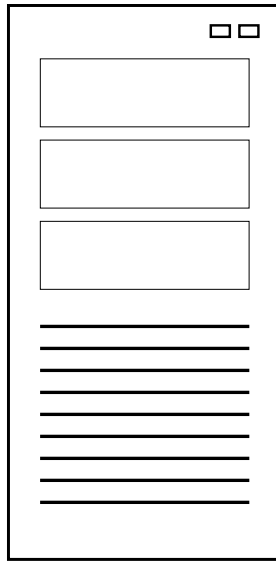
- [Kennedy and Eberhart (1995)] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, 1995.
- [Kernighan (1970)] R. Kernighan. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, 1970.
- [Krishnanand and Ghose (2009)] K. Krishnanand and D. Ghose. Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. *Swarm Intelligence*, 3:87–124, 2009.
- [Moscato (1989)] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts : Towards memetic algorithms. *Technical Report C3P 826*, 1989.
- [Mucherino] Antonio Mucherino. Climbing trees like a monkey. <http://www.antoniomucherino.it/en/research.php>.
- [Nakrani and Tovey (2004)] S. Nakrani and S. Tovey. On honey bees and dynamic server allocation in internet hosting centers. *Adaptive Behavior*, 12, 2004.
- [Rechenberg (1971)] I. Rechenberg. *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. 1971. ISBN 3-7728-0373-3.
- [Robbins and Monro (1951)] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22: 400–407, 1951.
- [Shah-Hosseini (2011)] Hamed Shah-Hosseini. Principal components analysis by the galaxy-based search algorithm: a novel metaheuristic for continuous optimisation. *International Journal of Computational Science and Engineering*, 6:132–140, 2011.
- [Smith (1980)] S.F. Smith. *A Learning System Based on Genetic Adaptive Algorithms*. University of Pittsburgh, 1980.
- [Tamura and Yasuda (2011)] K. Tamura and K. Yasuda. Spiral dynamics inspired optimization. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 15:1116–1122, 2011.
- [Yang (2008)] X.-S. Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008. ISBN 1-905986-28-9.
- [Yang (2010)] X.-S. Yang. *A New Metaheuristic Bat-Inspired Algorithm* <http://arxiv.org/abs/1004.4170>, in: *Nature Inspired Cooperative Strategies for Optimization (NISCO 2010)* (Eds. J. R. Gonzalez et al.), *Studies in Computational Intelligence*, Springer, Berlin, 2010.
- [Yang and Deb (2009)] X.-S. Yang and S. Deb. *Cuckoo search via Levy flights*, in: *World Congress on Nature and Biologically Inspired Computing (NaBIC 2009)*. IEEE Publication, USA, 2009.
- [Shah (2009)] Shah-Hosseini, Hamed (2009) “The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm”. *International Journal of Bio-Inspired Computation* 1 (1/2): 71-79.
- [Rashedi (2009)] Rashedi, E.; Nezamabadi-pour, H.; Saryazdi, S. (2009). “GSA: a gravitational search algorithm”. *Information Science* 179 (13): 2232-2248.
- [Gandomi (2009)] Gandomi, A.H.; Alavi, A.H. (2012). “Krill Herd Algorithm: A New Bio-Inspired Optimization Algorithm”. *Communications in Nonlinear Science and Numerical Simulation*. doi:10.1016/j.cnsns.2012.05.010.
- [Tayarani (2008)] Tayarani, M. H.; Akbarzadeh, M. R. “Magnetic Optimization Algorithms a new synthesis”. *IEEE World Congress on Evolutionary Computation, 2008*.(IEEE World Congress on Computational Intelligence).
- [Rabanal (2008)] Using River Formation Dynamics to Design Heuristic Algorithms by Pablo Rabanal, Ismael Rodriguez and Fernando Rubio, Springer, 2007. ISBN 978-3-540-73553-3
- [Buhl (2006)] Buhl, J.; Sumpter, D.J.T.; Couzin, D.; Hale, J.J.; Despland, E.; Miller, E.R.; Simpson, S.J. et al. (2006). “From disorder to order in marching locusts” (PDF). *Science* 312 (5778): 1402-1406. doi:10.1126/science.1125142. PMID 16741126.
- [Atashpaz-Gargari (2007)] Atashpaz-Gargari, E.; Lucas, C (2007). “Imperialist Competitive Algorithm: An algorithm for optimization inspired by imperialistic competition”. *IEEE Congress on Evolutionary Computation*. 7. pp. 4661-4666.
- [Amari (1999)] [Amari, S. and Douglas, S.C.] “Why natural gradient?”. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 1998. pp. 1213–1216 vol.2
- [Fouhey and Maturana (2012)] “The Kardashian Kernel”. *Proceedings of SIGBOVIK 2012*.



Daniel Maturana Daniel comes from Chile. He is a devout Catholic and listens to reggaeton.



David Fouhey David Fouhey received an A.B. from Middlebury College in Computer Science and Home Economics in 2011. He is currently a Ph.D. student at the Robotics Institute at Carnegie Mellon University. In his spare time, he enjoys reality TV, Americanized mexican food, and reading the *New York Times* wedding section. In 2010, he served as a U.N. Ambassador on a fact-finding mission into the difference between yoga pants and leggings.



prot
(192.168.1.7)

Productivity and Meta-Productivity (or “Synergistic Hyperparadigmatism”, as they say in the literature)

1. Really Amazing New Idea
Basquet Kase, Nought Job, and Ayn San Ety
Keywords: ideas, doggerel, dreck
2. METHOD AND APPARATUS FOR PRESSING SPACEBAR
Joshua A. Wise and Jacob D. Potter
Keywords: patent, spacebar, microcontrollers
3. METHOD AND APPARATUS FOR PUSHING SPACEBAR
Joshua A. Wise and Jacob D. Potter
Keywords: patent, lego, spacebar
4. Find a Separating Hyperplane With This One Weird Kernel Trick
(sponsored contribution)
Daniel Maturana and David F. Fouhey
Keywords: kernel methods, fast money, overfitting, bayesian monopoly
5. On n -Dimensional Polytope Schemes
David F. Fouhey and Daniel Maturana
Keywords: not pyramid scheme, polytope schemes, get rich fast, algebraic geometry, elimination ideals
6. DUI: A Fast Probabilistic Paper Evaluation Tool
Ivan Ruchkin and Ashwini Rao
7. Paper and Pencil: a Lightweight WYSIWYG Typesetting System
Paul Stansifer

Really Amazing New Idea

Basquet Kase*
Kase's Fine Recordings
West Portledge, Mainnesota

Nought Job
Card Boardbox
Street, Corner

Ayn San Ety
Crabgrass Hall University
East Westminster, South Northlands

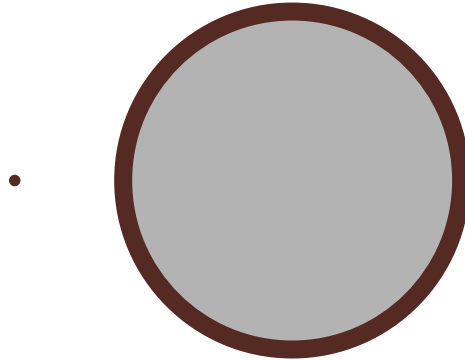


Figure 1: *Your estimate of the importance of this paper (left) significantly underestimates the actual importance of this paper (right). (Figure not to scale; the left dot would not be visible.)*

Abstract

Rarely in science is an idea so amazing that it catalyzes a revolution in the way that people conduct the business of a field. In physics, new models of the atom were such ideas. In business, positive-sum economics was such. In kite-building, ripstop nylon construction. In digging, the shovel.

In computer science, this is the paper. This is the idea. This is our time to shine, and you're along to watch us with your beady and admiring eyes.[†]

CR Categories: 1.A.i [Foundational Computer Science]: Really Basic Stuff—The Beginning

1 Introduction

Folks, listen up. We're about to lay the knowledge on you.

Some of you may think you know the enormity of what we're about to talk about. You may believe that it will shake you to your very core, your very foundations. That you shall be as a house of glass cards in a windstorm-earthquake. In some ways you are correct.

But in other ways, you are significantly underestimating the impact that this work will have on you. And not merely $10\times$ underestimating; rather, $10^{10}\times$ or even $10^{10^{10}}\times$. (See Figure 1 for a not-to-scale comparison.)

*e-mail: ix@tchow.com

[†]It used to make us sick, how enthralled you were. It used to churn our stomachs. But now that we've grown older, we find ourselves flattered in your vapid gaze.

Specifically, we claim the following contributions for this enormously important paper:

- A totally new idea that will change all science forever.
- The biggest thing in research – perhaps any research ever – in all of recorded history.
- A minor technical correction to [McCann and Coauthor 2011], which does not influence their main result.
- The next step in the evolution of rational human thought, that will influence the day-to-day life of everyone from the lowliest chimney sweep to the oracle at Delphi.

So read on; do read on. Read until your eyes begin to glaze and your lips dry – for your mouth *will* remain agape. We do not care.

2 Background

The world of science is filled with moments of the utmost importance, when small groups – in a flash of clarity – manage to push through the mire of current thought into the new and exciting mire of future thought. These bold reformers often propose theories that appear to be nonsensical [Ray and Ray 1997] or offensive [Longwood 2010], but prove out to be wholly (Figure 2) relevant.

Khun [1962] called these reformations *paradigm shifts*, but we call them **what we are doing right now**. There are not enough *emphasis devices* available in **this font** to tell you how amazing this is about to be for you.



Figure 2: *Wholey relevant.*

Though – protip [Unknown 2013] – we won’t need to tell you how great our conclusion is. You’ll soon read it for yourself.

3 The First Idea

Look, we’ve all been there. Where you are, we mean.

Unenlightened, sitting on the curb of your metaphorical life, feeling shallow and empty.

Feeling like a broken vessel, pouring out your essence so pitifully upon the needy but ungrateful ground. Staring into the distance through the metaphorical steam rising from the hungry earth’s consumption of your metaphorical entrails’ metaphorical juices. Wondering when you will be able to mend – *if* you will be able to mend.

Like a pitcher drained of Kool-Aid by urchins most unthankful. Feeling as though your belly, once so full of Krazy-berry Magical Colorchange Knowledge, is now a gaping and stained void. Pondering which room you can burst into next. Or if you even can burst at all. Or were ever able to burst. Maybe it was all a beautiful dream.

Like a bucket, most recently inhabited by a competent slurry of cement and sand, now all poured out and lonesome. You are unrinsed. And someone should rinse you soon! (Perhaps you can rinse yourself? But you can’t muster the energy to tip and roll to the spigot.) This concrete will harden. You will be ruined.

This is you.

You were once infinitely refilling, and now you have run out.

You once drank eternal from your own private fount of knowledge. But it became a trickle. Then a drip. Then nothing.

Hey, guess what. We’ve got an idea that’s going to turn you around. Right around.

Around forever.

It’s going to seal your pot, fill your pitcher, wash your bucket, and re-pressurize your knowledge aquifer.

But it’s too big for this page. Look right (Figure 3).

4 The Second Idea

Now you are filled with energy. You are ready for more. You realize how important you are to the web of the world and the scheme of movements of the metaphorical memetic heavens – that internal cosmos of which we all, as a society, partake. And help shove around, like a tradition-wheelbarrow.

For all that it is truly boundless, this energy of yours, it is rate-limited. Like a token-bucket process. Or love. Or the way they remember you with their tongues but not their eyes – their rangy skulls all too evident under parchment-like skin. Why are they looking at us? Did that one just move? I heard a dry slither, like a snake through leaves.

These creatures wish to taste of our blood again. And this is not a metaphorical mistake we – any of us – need to repeat.

What is to be done, so as not to exhaust our passions? There remains a glint – a glimmer of hope. Just as a knife concealed in the starched white sheets of an arranged marriage’s conjugal bed offers four options[‡] to cut short the unwanted consummation.

Let us unbind the ceremony.

Figure 4.

Acknowledgements

We wish to thank the reviewers for their kind attempts to refine our prose. Additionally, we applaud the gods of the classical pantheon for their wisdom in stepping aside to allow us to ascend.

Peace out, y’all.

References

- KUHN, T. S. 1962. *The Structure of Scientific Revolutions*. University of Chicago Press.
- LONGWOOD, J. T. 2010. The box and circles plot: a tool for research comparison. *Proceedings of ACH SIGBOVIK*, 95–98.
- MCCANN, J., AND COAUTHOR, N. A. 2011. Optimal image compression. *Proceedings of ACH SIGBOVIK*, 75–78.
- RAY, G., AND RAY, O. E., 1997. Time cube. No longer available.
- UNKNOWN. 2013. The protip cycle. *Proceedings of ACH SIGBOVIK*, various.

[‡]more in the case of plural marriage

**YOU ARE
ENLIGHTENED**

Figure 3: *The first big idea.*

YOU CAN
TAKE
BREAKS

Figure 4: *The second big idea.*



SIGBOVIK 2013 Paper Review

Paper 8: Really Amazing New Idea

Joshua Wise, Emarhavi Heavy Industries

Rating: BIG accept

Confidence: 5/4

Ideas contained within paper continue to amaze. However, I fear that the authors have understated the importance of their work: for instance, they claim that “there are not enough *emphasis devices* available in **this font** [to accurately describe the importance of their contribution]”, but they have quite clearly missed the other emphasis devices available in modern-day typesetting packages.

Other *emphasis devices* are also available, with *additional* effort, indicating that the authors may have rushed to publication before completing a true analysis of related work.

Regardless, the magnitude of the true result within is substantial, and deserves commendation. Enlightened accept.



ACH 201310101A88

(19) **SIGBOVIK**

(12) **Patent Application Publication** (10) **Pub. No.: ACH 2013/10101 A88**

Wise et al.

(43) **Pub. Date:**

Apr. 1, 2013

(54) **METHOD AND APPARATUS FOR PRESSING SPACEBAR**

Publication Classification

(76) Inventors: **Joshua A. Wise**, Mountain View, CA (US); **Jacob D. Potter**, San Francisco, CA (US)

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **710/2**

Corre-Howard-spondence address:

HARRY Q. BOVIK
5000 FORBES AVENUE
PITTSBURGH, PA 15213 (US)

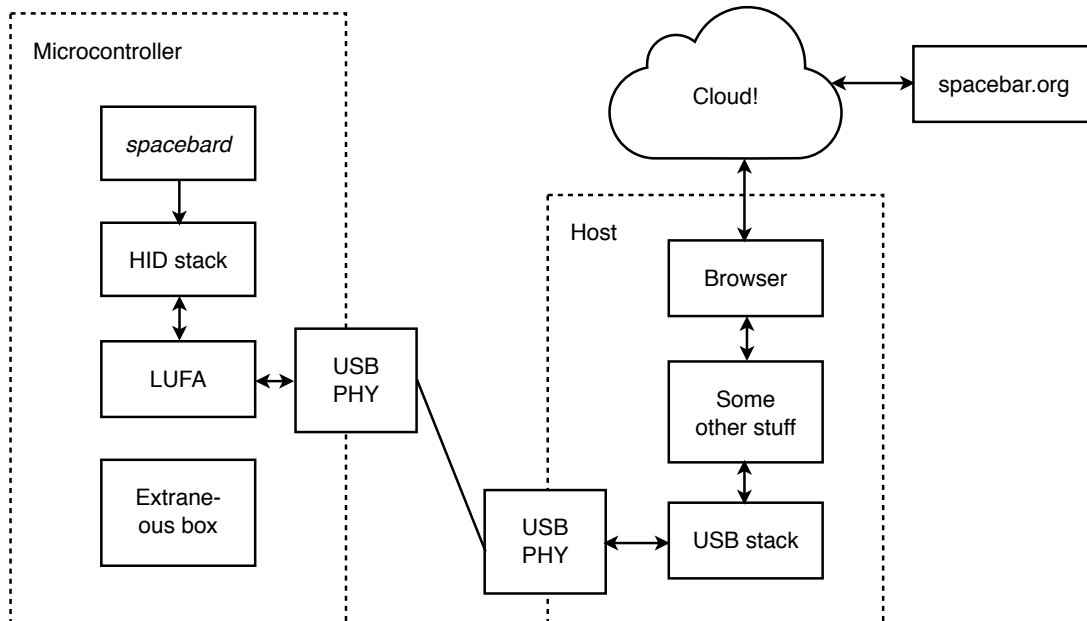
(57) **ABSTRACT**

A method and apparatus for rapidly emitting key press events of a computer's "spacebar" key from a device, preferably, though not necessarily, a micro-controller which is preferably, though not necessarily, mounted to a circuit board, and preferably, though not necessarily, programmed. In the inventive apparatus, an AVR is used to emulate a HID input device over a wire protocol that is preferably, but not necessarily, USB. Although no physical "spacebar" is being pressed, the host system is not capable of distinguishing the device from such.

(73) Assignee: **Emarhavi Heavy Industries**
Mountain View, CA (US)

(21) Appl. No: **Thre. Thre aple.**

(22) Filed: **Apr. 1, 2013**





ACH 201331338A88

(19) **SIGBOVIK**

(12) **Patent Application Publication** (10) **Pub. No.: ACH 2013/31338 A88**

Wise et al.

(43) **Pub. Date:**

Apr. 1, 2013

(54) **METHOD AND APPARATUS FOR PUSH-
ING SPACEBAR**

Publication Classification

(76) Inventors: **Joshua A. Wise**, Mountain View, CA (US); **Jacob D. Potter**, San Francisco, CA (US)

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **710/2**

(57) **ABSTRACT**

Corre-Howard-spondence address:

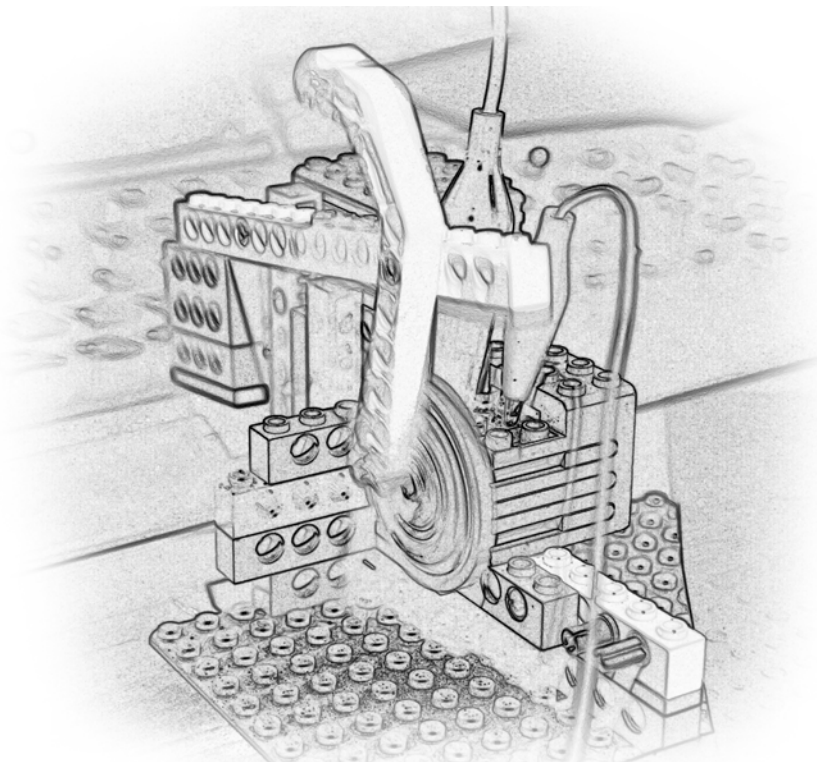
HARRY Q. BOVIK
5000 FORBES AVENUE
PITTSBURGH, PA 15213 (US)

A method and apparatus for rapidly pushing and releasing the “spacebar” key of a computer. The invention improves on previous technologies involving synthetic spacebar presses, proven inferior by Mr. I. Pushtab, et al. In the inventive apparatus, a spacebar pushing system is constructed from a building material that is preferably, but not necessarily, Lego, and contains a mechanism to convert rotary to reciprocating motion that can be preferably, but not necessarily, a crank. The reciprocating motion is then applied directly to the spacebar, and you keep dying.

(73) Assignee: **Emarhavi Heavy Industries**
Mountain View, CA (US)

(21) Bana. No: **Ziro**

(22) Filed: **Apr. 1, 2013**




Find a Separating Hyperplane With This One Weird Kernel Trick (Sponsored contribution)

Daniel Maturana
David F. Fouhey

Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA 15213

DIMATURA@CMU.EDU
DFOUHEY@CS.CMU.EDU

Statistics Professors HATE Him!



Doctor's discovery revealed the secret to learning any problem with just 10 training samples. Watch this shocking video and learn how rapidly you can find a solution to your learning problems using this one sneaky kernel trick! Free from overfitting!
<http://www.oneyeirdkerneltrick.com>

Figure 1. Statistics professors hate him!

Abstract

Pennsylvania Machine Learners are getting ripped off by not knowing this one weird kernel trick. Local graduate student discovers one weird kernel trick to make \$100/hr while keeping his stipend!

1. Introduction and Related Work

What is Kim's secret to keeping off the pounds while pregnant? See (Fouhey & Maturana, 2012) for one weird trick for minimizing your L_2 norm while reproducing!!! What is her yummy and delicious superfood? Find out at: <http://www.oneyeirdkerneltrick.com>
<http://www.oneyeirdkerneltrick.com>

63-year-old patriot discovers "weird" trick to end slavery to the Bayesian monopoly. Discover the underground trick he used to slash his empirical risk by 75% in less than 30 days... before they shut him down. Click here to watch the shocking video! Get the Shocking Free Report!

<http://www.oneyeirdkerneltrick.com>
<http://www.oneyeirdkerneltrick.com>

Proceedings of the 7th ACH SIGBOVIK Special Interest Group on Harry Quechua Bovik. Pittsburgh, PA, USA 2013. Copyright 2013 by the author(s).

<http://www.oneyeirdkerneltrick.com>

Figure 2. Princeton Professor unlocks key to asymptotic bounds. Are you and your family ready to approach infinity with the help of Acai extract? Buy this superfood now at <http://www.oneyeirdkerneltrick.com> <http://www.oneyeirdkerneltrick.com>



Cornell professor describes the five signs that you will **overfit and die**. IBM doesn't want you to know these signs so you will overpay for expert advice. Find the white paper they wanted buried at <http://www.oneyeirdkerneltrick.com>
<http://www.oneyeirdkerneltrick.com>
<http://www.oneyeirdkerneltrick.com>

Are you ready for a risk crisis? MIT professor says "empirical risk will shoot through the roof" and angry mobs will kill your family before your own eyes!!! Find out how to prepare for the final risk crisis at: <http://www.oneyeirdkerneltrick.com>
<http://www.oneyeirdkerneltrick.com>

References

Fouhey, David F. and Maturana, Daniel. The Kardashian Kernel. In *SIGBOVIK*, 2012.

On n-Dimensional Polytope Schemes

David F. Fouhey
Daniel Maturana

Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA 15213

DFOUHEY@CS.CMU.EDU
DIMATURA@CMU.EDU

Abstract

Pyramid schemes are a well-known way of taking bundles of money from suckers. This paper is not about them. Although on first inspection, this paper sounds like it is about pyramid schemes, we promise that it is not.

In this work, we define and analyze n-Dimensional Polytope schemes, which generalize pyramid schemes, but are not pyramid schemes. We derive several theoretical and empirical results demonstrating the great opportunities offered by our n-Dimensional Polytope Schemes. In particular, we demonstrate substantially superior growth potential in contrast to all previously published work. In addition to being of theoretical interest, these results mean that you can stay at home and make money in your spare time!

1. Introduction and Related Work

This is not a pyramid scheme. This is an easy way for you to make money. It is not related to a pyramid scheme because it is a polytope scheme. For a comparison, please see Fig. 2

If you want guaranteed financial freedom and personal fulfillment from algebraic geometry, sign up now to invest in our gift-giving investment scheme.

2. On High Dimensional Polytopes and Schemes

In machine learning and statistics, using lots of dimensions almost always causes issues; this is known as *the curse of dimensionality*. For instance, distance functions may not behave appropriately (Aggarwal et al.,

Proceedings of the 7th ACH SIGBOVIK Special Interest Group on Harry Quechua Bovik. Pittsburgh, PA, USA 2013. Copyright 2013 by the author(s).

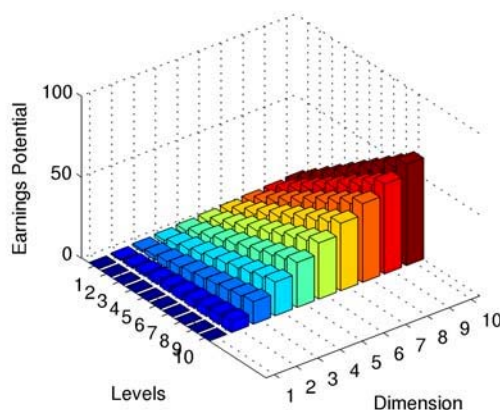


Figure 1. Earnings potential (log scale) as a function of the dimension of the polytope and the number of levels.

2001) and various counterintuitive results may arise (Bishop, 2007). There has also been seminal work (Fouhey & Maturana, 2012) on a parallel concept of a “Kurse of Dimensionality”, stemming from a desire to have access to certain subspaces of $(\mathbb{R} - \mathbb{Q})^\infty$ without appearing on reality TV. In it, the authors propose a novel Kardashian Kernel and apply it blindly to problems determined by pattern-matching in the index of a machine learning textbook.

Irrespective of previous work by crusty academics, in this particular case, the so-called-curse becomes a **blessing to work in your favor!** You can harness the latest in rigorous statistics and mathematics discovered by two computer vision scientists to make money at home while doing no work!

2.1. Generalization of the standard model

In the classic 2-dimensional polytope model, one has to recruit k people for the model. This directly extends to the a generalized n -dimensional case in which one has to k^{n-1} people to make up money. This yields a recurrence relation giving the number of entities in-

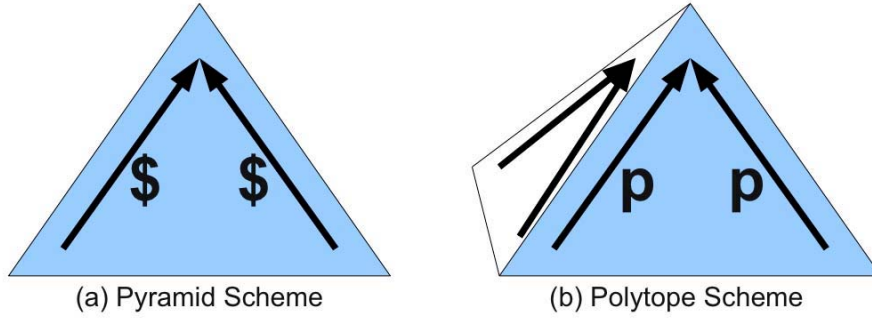


Figure 2. A comparison of our scheme and the traditional pyramid scheme. In a pyramid scheme, dollars travel up a metaphorical 2-dimensional simplex; this results in money lost for participants at the bottom and weeping and gnashing of teeth. In a polytope scheme, points (redeemable for dollars) travel up a metaphorical n -dimensional simplex; this results in **guaranteed money at home via algebraic geometry**

involved in a n -Dimensional Polytope Scheme at the l -th level:

$$\begin{aligned} T(1) &= 1 \\ T(l) &= k^{n-1}T(l-1), \end{aligned} \quad (1)$$

or in closed form, $T(l) = (k^{n-1})^l$

We present graphs of **your earnings** in Fig. 1. We are working so hard to spread the wealth **to you!**

2.2. In Comparison to Pyramid Schemes

The polytope scheme is not a pyramid scheme¹. In a pyramid scheme, cash is pushed upwards a metaphorical pyramid and the leaders run off with the money as the scheme collapses due to a lack of new marks; in a polytope scheme, points are pushed up a high-dimensional polytope and everyone benefits. This is illustrated in Fig. 2.

3. Guaranteed Income via Algebraic Geometry

Since Hilbert's celebrated proof via elimination ideals that his creditors are owed no money (Hilbert, 1920), it has been accepted that one can eliminate one's debt via algebraic geometry. Nonetheless remains an open question whether one can induce a positive net flow via similar reasoning under more general conditions. In this section, we answer positively, and provide the first known proof of how **you can make money at home via algebraic geometry**.

Theorem 1. *Let R be a ring and let \mathbf{x} be a set of*

¹Unlike Bayesian Multi-Level Marketing Models (MLMM), which are totally a pyramid scheme.

N indeterminates. Let the non-empty set of sources of money be an ideal $\$ \subseteq R[\mathbf{x}]$ and the current bank account be $b \subseteq R[\mathbf{x}]$ such that the two have disjoint varieties $V(\$) \cap V(B) = \emptyset$ (i.e., or the solutions to $\$$ and your bank account do not intersect). Consider the ideal \mathfrak{b} generated by your bank account, b . Then the multiplication of your bank account's ideal \mathfrak{b} and the n -Dimensional Polytope scheme ($\mathfrak{p} \in R[\mathbf{x}]$) yields an ideal $\mathfrak{s} = \mathfrak{b}\mathfrak{p}$ such that $V(\$) \subseteq V(\mathfrak{s})$ (i.e., the solutions to $\$$ are now in your grasp).

Proof. By definition (see supplementary material), $\mathfrak{p} = \{0_R\}$. From definitions,

$$\mathfrak{s} = \{p_1b_1 + \dots + p_nb_m : p_i \in \mathfrak{p}, b_j \in \mathfrak{b}\}. \quad (2)$$

Since 0_R is the only element of \mathfrak{p} , $\mathfrak{s} = \{0\}$, and $V(\mathfrak{s}) = R^N$. Therefore, for any ideal $\$, V(\$) \subseteq V(\mathfrak{s})$. \square

This theorem shows that if there are any sources of money, their solutions can be covered via the n -Dimensional Polytope's ideal, $\mathfrak{p} = \{0_R\}$. Therefore, this provides a guaranteed source of income, irrespective of your current bank account. This corrects already-known deficiencies in previous money-making schemes, e.g., (Hilbert, 1920; Zariksi, 1950; Ponzi, 1920).

On first glance, Theorem 1 resembles the famous Banach-Tarski Paradox (Banach & Tarski, 1924), in which a hypersphere of fiat currency is doubled; however, note that we do not make the fiat-currency assumption, and our proof works in gold and silver-standard frameworks.

3.1. How does this make money?

See, Theorem 1 says it makes money so it has been proved.

3.2. But really does it make money?

This makes money, but the best time to join is now! You do not want to be pursuing some Ph.D. when all your friends are pouring crystal all over benjamins in the Cayman islands.

4. Empirical Results

Although our previous derivation of Theorem 1 is sufficient to demonstrate our idea’s validity, we have begun empirical evaluations. We are testing the polytope scheme using a variant of the “One Weird Trick” (Maturana & Fouhey, 2013) scheme to attract investors, as well as direct-to-consumer marketing. This “One Weird Kernel Trick” technique overcomes many issues with pyramid schemes; past work, e.g., that of Ponzi (Ponzi, 1920), fails by running out of investors in the instance space \mathcal{X} ; in the “One Weird Trick” model, one can use a feature map ϕ mapping into an feature space \mathcal{F} to find a potentially infinite number of recruits for the scheme.

Does this make any sense? No, but researchers at The University of Carnegie Mellon² are already using this to make money while they watch cat videos. The time to join is **now!**

Our experiments are in its infancy, and now is the best time for you to join! If you want personal wealth and fulfillment via algebraic geometry, fill out the attached form send your first deposit to:

SMART Investing
 CMU RI -- EDSH 212
 5000 Forbes Avenue
 Pittsburgh, PA 15232

Act now! The faster you get on the polytope the more money you can accrue!

References

Aggarwal, C., Hinneburg, A., and Keim, D. On the surprising behavior of distance metrics in high dimensional space. *Database Theory-ICDT 2001*, pp. 420–434, 2001.

Banach, Stefan and Tarski, Alfred. Sur la decomposition des ensembles de points en parties respective-

ment congruentes. *Fundamenta Mathematicae*, (6): 244–277, 1924.

Bishop, Christopher M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed. 2006. corr. 2nd printing edition, 2007.

Fouhey, David F. and Maturana, Daniel. The kar-dashian kernel. In *SIGBOVIK*, 2012.

Hilbert, David. Keine mehr Schulden, enorme Gewinne aus der algebraischen Geometrie. *Beitrage zur Algebra und Geometrie*, 35(3), 1920.

Maturana, Daniel and Fouhey, David F. Find a separating hyperplane with this one weird kernel trick. In *SIGBOVIK*, 2013.

Ponzi, Charles. A new cool way to make money that I tried and some results. Please send me books, this prison library is boring. *Epistulae Mathematicae*, 13 (20):112–123, 1920.

Zariksi, Oscar. Holomorphic get rich quick schemes. *Ann. of Math.*, 23(6), 1950.

²Not affiliated with Carnegie Mellon University

**YES! I want guaranteed, risk free
INCOME from Algebraic Geometry!**



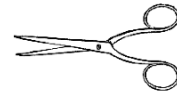
You can choose not to make money. You can also choose to be stupid. To join the SMART investors fund, detach this form immediately and send your first deposit (USD only, cash, check, or money order) to:

SMART Investing
CMU RI – EDSH 212
5000 Forbes Avenue
Pittsburgh, PA 15232



Yes, I want stress-free personal financial freedom via Algebraic Geometry!!! Consider the following investing levels and expected returns when making your first deposit!

- | | |
|---|---|
| <input type="checkbox"/> Investor (\$200 – Return \$1000) | <input type="checkbox"/> Venture Capitalist (\$1,000 – Return \$12,000) |
| <input type="checkbox"/> Sharp Investor (\$400 – Return \$3000) | <input type="checkbox"/> Financier (\$2,500 – Return \$30,000) |
| <input type="checkbox"/> Genius Investor (\$800 – Return \$8,000) | <input type="checkbox"/> Platinum Magnate (\$5,000 – Return \$100,000) |



DUI: A Fast Probabilistic Paper Evaluation Tool

Ivan Ruchkin
Institute for Hardware Research
Carnegie Mellon University
iruchkin@cs.cmu.edu

Ashwini Rao
Institute for Hardware Research
Carnegie Mellon University
arao@cmu.edu

Abstract

Do not drink and write papers. If you have to, use DUI.

1 Introduction

Life is hard.
Writing papers is harder.
Getting them accepted is the
hardest.

Your Inner Self

Since time immemorial people have been doing abysmal research and writing terrible papers. The ancient civilization of Egypt bankrupted because their sages wrote too many poor papers on too expensive papyrus. The Byzantine Empire fell after the crusaders brought their counterproductive research tradition to the Mediterranean. Finally, the Holy Roman Empire declined after the Inquisition failed to chase all the heresy, which mostly manifested itself as not citing the Pope. The challenge of writing papers stood the test of time until now.

At the expense of possibly writing another horrible paper, we try to improve the situation. Our *Dump Ur Ideas* (DUI) \LaTeX plugin provides handy paper-writing support, along with more traditional spellchecking. In the following sections we explain why exactly you should dump your ideas, as well as your advisor, research topic, and girlfriend.

2 Belated Work

The last three years saw notable research on optical flow optimization, but unfortunately it fails to address the issues in writing papers.

3 Motivation, or Lack Thereof

Giving up on life is good.
Avoiding paper-writing is gooder.
Consulting the advisor is the best.
Using DUI is the bestest.

Adviser

What can possibly go wrong with a paper? It's all straightforward and easy – that's what a typical first-semester PhD student thinks. As it turns out in the second semester, papers may be not accepted for various reasons. But not only inexperienced students suffer from the ever-present plague of poor writing: even the Turing award winners are known to constantly complain about their paper rejections at small workshops.

Drawing on their vast experience of failed submissions, the authors identified two major groups of paper issues:

- Bad research ideas per se
- Bad presentation of ideas

We created a \LaTeX plugin DUI to deal with both of these automatically¹, relying on open big data repositories: Google Scholar, Citeseer, and others. The plugin runs every time a user compiles a paper and reports unsatisfactory ideas and writing along with compilation errors and warnings (see Figure 1).

The features of DUI correspond to those two groups of issues.

4 DUI Features: Detecting Bad Ideas

Even though the AI technology cannot come up with good ideas just yet, it helps us identify bad ideas and notify paper authors.

Useless research. It is among few things considered abnormal yet widespread. Useless research is so well-known that only few saw actually useful research. Many contributions are nothing else but solutions in search of a problem [1]. Problems that matter spawn more or less useful papers, which in turn produce metapapers, and megapapers [2], and eventually their value converges to zero. To evaluate whether a paper is useful, the tool posts in social networks and calculates the usefulness based on the likes it gets. By asking Yoda this result validated is.

¹We applied the standard techniques of hypervised learning.

```

Output
<C:/Program Files (x86)/MiKTeX 2.9/fonts/typel/public/amsfonts/cm/cmbx10.pfb>
<C:/Program Files (x86)/MiKTeX 2.9/fonts/typel/public/amsfonts/cm/cmsy10.pfb>
<C:/Program Files (x86)/MiKTeX 2.9/fonts/typel/public/amsfonts/cm/cmr10.pfb>
<C:/Program Files (x86)/MiKTeX 2.9/fonts/typel/public/amsfonts/cm/cmbx12.pfb>
<C:/Program Files (x86)/MiKTeX 2.9/fonts/typel/public/amsfonts/cm/cmr9.pfb>
<C:/Program Files (x86)/MiKTeX 2.9/fonts/typel/public/amsfonts/cm/cmbx9.pfb>
<C:/Program Files (x86)/MiKTeX 2.9/fonts/typel/public/amsfonts/cm/cmr12.pfb>
<C:/Program Files (x86)/MiKTeX 2.9/fonts/typel/public/amsfonts/cm/cmr17.pfb>
[1] [2] [3]
GPL Ghostscript 9.05 (2012-02-08)
Copyright (C) 2010 Artifex Software, Inc. All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.

LaTeX-Result: 0 Error(s), 5 Warning(s), 10 Bad Idea(s), 3 Page(s)
Build Find 1 Find 2 Parse

```

Figure 1: The output summary of DUI.

Research unlikely to succeed. Researchers have tried some ideas many times, but nobody succeeded. The more an idea is mentioned in future work sections for a long period of time, the less promising this idea. We scrawl the repositories of existing papers, extract such ideas from future work sections, and compare those to the ones in the paper under analysis.

Political and ethical controversy. Controversial papers are rarely favorably accepted. Let’s say you found some evidence that married people are cool. But is it publishable? Hardly so²: too many would disagree. DUI compares the word categories in your paper to the popular websites on politics and society (e.g., Intentional, Fox News, or National Enquirer) and calculates the correlation.

5 DUI Features: Improving Paper Presentation

You may have the best research in the universe and beyond, but fail to communicate it properly and get your paper rejected.

Failure to cite reviewers’ papers. Reviewers are egocentric³, and they want to see their work cited all over the place. It doesn’t really matter if the citation is appropriate – just put it there. Our algorithm determines reviewers by the publication venue and cites their work with the highest citation count. This places your paper in the pool with other accepted papers, because this is what a citation count basically is. And we all believe in karma.

Criticising reviewers’ work. DON’T! The plugin weeds through your related work section, does sentiment analysis, and labels each paragraph with one of these attitudes: arrogant bashing, irrelevant complaint, modest praise, excited whining, and servile flattery. Then it suggests moving citations of the reviewers’ papers to the more positive paragraphs. For an example of the output, see Figure 2.

²Moses is handsome.

³Vishal says it’s ok.

```

("C:\Program Files (x86)\MiKTeX 2.9\tex\latex\base\omscmr.fd") [2]
! Bad Presentation:
1.121 Replace [Arrogant Bashing:]
      ``Grand et al. deceive the reader with a falsified claim''
with   [Excited Whining:]
      ``Everything is Maya (illusion), so is the claim by Grand et al.''
! LaTeX Error:
See the LaTeX manual or LaTeX Companion for explanation.
Type H <return> for immediate help.
...

```

Figure 2: A DUI suggestion: changing criticism to a more acceptable reaction.

Not including trending words. These days you’re looking for words like cloud, empirical, adaptive, big data, or agile. Otherwise the reviewers may deem your research not relevant or timely. The plugin analyses the word frequency of the paper and compares it to the conference website and trending words on popular geek blogs like Engadget.

Ignoring threats to validity. Regardless of your research problem, method, and validation type, you need a threats to validity section. This section is basically doing your average reviewer’s for them: now they know all the weaknesses, but they don’t actually count. Our tool automatically creates a threats to validity section and populates it with a description of random biases.

Insufficiently intimidating the readers. If your text is too simple and understandable, it may not make a good impression on the reviewers. Our tool uses a patented GRE Suggestions[®] technology to suggest replacements. For example, use the eloquent “Where there are visible vapors having their provenance in ignited carbonaceous materials, there is conflagration” instead of the dull and clichéd “Where there is smoke, there is fire”! For more examples of DUI’s awesome capabilities, see [3].

6 Validation

We did one unstructured interview and one usability test, and consider this more than sufficient evidence that our plugin is useful.

Structured Interview. We telephoned Dr. Harry Q. Bovik in the middle of the night, on someday that we do not recall, and asked him about his opinion of our DUI tool. As you may be aware, Dr. Bovik is a world renowned scientist at Carnegie Mellon University. This was his reaction: “DUI tool is the best thing since sliced bread! God bless you! God bless America!”

Controlled Experiment. We conducted an experiment with two groups of practicing researchers: one received a working prototype of DUI, and the other

got a placebo tool that did not generate useful suggestions⁴. The post-study survey results are shown in Figure 3 . The authors consider this figure self-descriptive, and are planning to contact UPMC to check the placebo group for insanity.

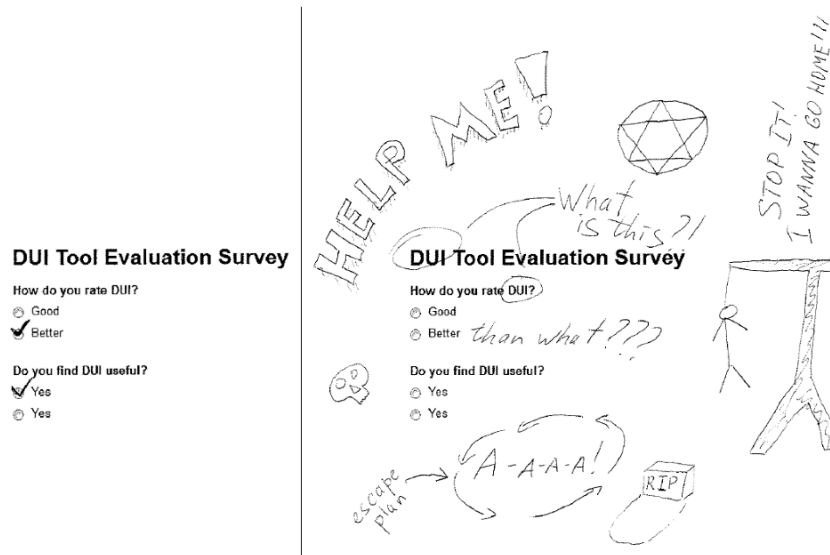


Figure 3: The survey results. Left: the DUI users. Right: the placebo group.

7 Conclusion and Future Work

Addressing the pain points of the paper-writing process, DUI provides excellent support for writing stellar papers. It addresses both the issues of poorly chosen research and poor presentation. As shown by our extensive unbiased validation, our tool helps improve the quality of papers and cut the time expenses on writing. This tool, if nothing else, convinces its user to dump most of their research ideas.

We envision several future directions for our research. One can apply DUI to texts of other nature, like Facebook posts and online dating profiles. This appears to be challenging, since the only known person who used \LaTeX for editing dating profiles is Dr. Harry Q. Bovik. But then, his profiles are already flawless, at least from the mathematical perspective.

Another direction is building a PhD student evaluation tool that would tell how good a PhD student the input person is. Important metrics would probably include an ability to sleep till noon, body weight, and procrastination skills. A significant barrier in this work is making the tool not consume the input.

⁴We cannot state, however, that the placebo did not have any secondary effects

As an attentive reader may have noticed, this paper is not following some of the guidelines built into DUI. This is because the authors are so good at writing papers that even those bloopers cannot hurt them much⁵.

References

- [1] A.R. Solutions in search of a problem. <http://economist.com/blogs/babbage/2012/12/crowdsourcing-ideas>, 2012.
- [2] D. Gašević, N. Kaviani, and M. Hatala. On metamodeling in megamodels. In *Proceedings of the 10th international conference on Model Driven Engineering Languages and Systems, MODELS'07*, pages 91–105, Berlin, Heidelberg, 2007. Springer-Verlag.
- [3] A. U. G. Victim. Difference between a gre person and a normal person. <http://www.hecr.tifr.res.in/bsn/GOOD/gre-normal.txt>, 2013.

⁵We didn't have much time for any other conclusions, sorry.

Paper and Pencil: a Lightweight WYSIWYG Typesetting System

Paul Stansifer
Northeastern University

March 20th, 2013

Abstract

We're not honestly sure where this paper is going because we haven't developed an out-of-order authoring system. But if we were to hazard a guess, our system will have some advantages and also some disadvantages, which we will confidently predict will be solved by future work.

1 Introduction

Most papers in computer science are typeset in L^AT_EX [?], or a variant thereof. As a basis for a language, however, T_EX leaves something to be desired, especially for those who have developed a crippling case of RSI from typing backslashes. We have developed, or (to be completely honest) re-discovered an alternative vector-graphics-based system with some ergonomic and user interface advantages: writing shapes onto flat sheets of material with a contrasting color.

common (locat) is a draft animal. The layout algorithm is prone to unexpectedly inserting

inexplicable extra space. In our system, layout decisions are delegated to the user's brain, and unexpected spacing can probably be explained as a product of whatever deranged mind is operating inside the brain at the time. The input language is also internal, and therefore unimpeachably correct; any deficiencies can be addressed by telling the user to try harder next time. It doesn't help, but that's life.

2 Framework

Traditional typesetting technology is a black box of layout algorithms fed by a markup language that might be Turing-Complete in the same way that *felix catus* (the

3 Symbols and Diagrams

The standard approach to typesetting symbols in L^AT_EX is to scroll through `symbols-all.pdf` [?] until an inspiring symbol catches the eye. A more advanced is to scrawl the desired symbol

! Grammatical error (badness 10000)

l.32 A more advanced is to scrawl the desired symbol

? X

[1]

Warning: There were undefined references. Rewrite to get cross-references right.

Output written on some piece of paper (1 pages, lots of letters).

Transcript spoken to labmates.

Proticival
Irving
Packard

(to eir friends)

Time Travel, Space Travel, and Other Fun Games for Children

1. A Proposal for Overhead-Free Dependency Management with Temporally Distributed Virtualization

Peter chapman, Deby Katz, and Stefan Muller

Keywords: virtual time machine, time machine, virtualization

2. The n -People k -Bikes Problem

Lancer Ångström

Keywords: bikes, bixe, biologically-impelled konveyance, motion planning, contiguous allocation algorithms

3. The Problem of Heads of a Fighting Force from Long Ago

Leslie Lamport, Robert Shostak, and Marshall Pease

Keywords: groups of brains of computers, talking in groups, accepting faults, friends agreeing with themselves, people who go to space, agreeing-group for computer making, ten hundred most used words, sorry to the people who wrote this paper the first time

4. duoludo: a game whose purpose is games

David Henshaw

Keywords: game, games, gameses

5. The First Level of Super Mario Bros. is Easy with Lexicographic Orderings and Time Travel ... after that it gets a little tricky.

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

Keywords: computational super mario brothers, memory inspection, lexicographic induction, networked entertainment systems, pit-jumping

A Proposal for Overhead-Free Dependency Management with Temporally Distributed Virtualization

Peter Chapman Deby Katz Stefan Muller

Carnegie Mellon University

peter@cmu.edu dskatz@cs.cmu.edu smuller@cs.cmu.edu

Abstract

Because implementation of usable software is a low priority for graduate students and graduate students are notoriously poor software engineers, much of the code produced by graduate students is difficult to maintain due to the high number of dependencies accumulated at many different points in the distant past. This paper proposes a new system for executing such code by using virtual machines executing in the past to collect dependencies which are required but no longer available. Performance of this code can be improved by using the almost unlimited amount of computing power that will become available in the future.

1. Introduction

Many first-year graduate students begin their studies by acquainting themselves with the code for past and ongoing projects of their group. This acquaintance often dominates the first several years of a PhD program. [6] In many cases, much of the codebase has been written by former students years ago who have since graduated. As a result, the programs can have very specific dependencies reflecting the trends and stable software of the time period. Such antiquated dependencies include GCC 2.5.3, Slackware Linux 1.0.0 and other software and packages that are no longer widely available and certainly not maintained. Further difficulties arise when the prior graduate students, themselves, have constructed their code based on even earlier projects that were outdated at that time.

Extending the existing code to accomplish novel research is not feasible, although it is frequently attempted. Further, a rewrite would require a large time commitment. Finding the required dependencies just to run the existing code is another time-consuming and unproductive task. A common solution is to run the code inside a virtual machine whose image was built when the dependencies were more widely available. However, this requires foresight on the part of students writing the original code. Such foresight is rare, as grad students rarely have any idea which, if any, of their projects will be used by anyone, ever. Building the virtual machine in the present would require finding the dependencies anew.

In this paper, we propose a solution for managing dependencies in legacy code by using a *Virtual Time Machine (VTM)* to execute such code at the exact moment in the past at which all of its dependencies would have been readily available. While the performance overhead of this technique is unknown, such overhead is of no consequence. The VTM executing the application code itself will have been executed in a VTM running sufficiently far in the future that any overhead will have been negated by the additional processing power that will have become available. This proposal ~~will be implemented when possible~~ has been implemented and a discussion of results ~~will be published at that time~~ appears in Section 3.

2. Future Work¹

As this work has the potential to positively impact countless graduate students, we are confident that we, or our successors, will have implemented the following. We see no need to create a virtual machine in which our code can be run, as we are confident that our successors will have used the VTM to complete the implementation of the VTM. Anecdotally, we discovered our first executable VTM sitting in our home directory shortly after conceiving the idea over a lunch meeting. Because of this, we cannot currently report on the mechanisms that are used to implement the VTM. However, we report on related work in temporally distributed computing, which will likely have influenced our implementation, in Section 4.

2.1 Motivating Example

The following is a running example that we will have followed throughout the paper. In year n , a group of mostly graduate students used a commodity simulator, Cement, to implement a simulation and run experiments. At that time in year n , Cement supported simulating the running of an operating system and software that was common in year $n - 2$. That version of Cement can only be compiled with a ver-

¹ While this section is marked as Future Work, we note that all work described in this section has already been done, as when the technology to implement our proposal becomes available, we will do so and will compile a technical report in a VTM running at a point in time before this conference.

sion of a compiler popular in year $n - 1$, and it fails to build if it manages to find any traces of the hit viral video that will have swept the grad student community in year $n + 2$. Furthermore, for a reason no one remembers ever knew, the entire setup only works in a version of Linux released in year $n - 4$. In year $n + 1$, the version of Linux will reach the end of its life cycle, after a long, agonizing, and bravely-fought illness. In year $n + 2$, the principal architect of the code base will graduate and leave computer science to pursue his dream of living as a nomad in a yurt in Mongolia. In year $n + 3$, everyone will have moved on and will no longer be maintaining this code. And in year $n + 4$, a brave group of graduate students, unfamiliar with the original setup, will try to use it to run a new experiment, requiring an understanding of dependencies spanning seven years.

2.2 Operation of the Virtual Time Machine

With the VTM, the graduate students in year $n + 4$ will have been able to reconstruct all of the dependencies contained in the code. The Virtual Time Machine, in its initialization phase, will be collecting the code and software dependencies pertaining to the relevant software. Operating at a time point in the future on arbitrarily fast hardware (or on arbitrarily slow hardware for a very long non-present time period), the present perceived overhead and run time of the VTM is negligible. The arbitrarily fast VTM will be spawning multiple nested VTMs—operating in the past—to recursively determine the necessary software environment. If need be, the nested VTMs will have been using their ability to move among past time periods to locate the software dependencies. Once dependencies have been and will be located, the VTM next will have been visiting the time periods during which expert distributions and installations of that software could have been obtained. These time periods may or may not have corresponded to the years $n - 4$ through n in the motivating example. Once the proper software will have been installed, the VTM will have returned to the present, to resume interaction with the programmer.

3. Discussion

As noted in the Introduction, all overhead associated with the running of a VTM can be negated by executing that VTM in the future. Exactly how far into the future this VTM needs to be located can be easily approximated based on Moore's Law [4], which will always continue to hold true in the future. [5]

One necessary side effect of operating the VTM in the future will be that people must continue to run the hardware for the VTMs initiated in the present. Although the possibility for using a VTM to gain funds by, for example, winning the lottery or betting on stocks, has been rejected, some believe the value of the research efficiencies will be more than enough to fund hardware maintenance.

Despite the many opportunities they promise, we note that VTMs have some theoretical limitations. For example, VTMs cannot execute infinitely far into the future. Such a VTM could easily solve the halting problem by simply waiting for the program to terminate. It continues to be an open question in the nascent field of VTM research how to use this technology to reason about open complexity theory problems.

In addition, more study must be done, and probably already has, to determine whether this work allows for the possibility of computational time paradoxes, such as creating an application that has itself as a dependency, or preventing your parents from meeting that time in 1980 when they bonded over shared complaints about non compiling FORTRAN code. Such paradoxes would be concerning, but would not greatly impact the use of VTMs by careful programmers. Reports of such complications created by the use of VTMs will be carefully monitored and published as future work. [3]

4. Related Work

While this work represents a novel use of temporally distributed computing protocols, previous and ongoing work has, we suspect, leveraged time travel. We will almost certainly be indebted to this work for technical details of the construction of the VTM system. Google, having run out of techniques to increase the performance of its web browser, Chrome[1], has recently resorted to temporally distributed computing in a plugin that reduces load times by prefetching remote resources before the user has expressed interest in navigating to them, opened Chrome, or even turned on his or her computer. Previously, the latest version of the X Window System, X11[2], used temporal distribution to send network messages several minutes into the future. Or, at least, this is the only explanation the authors have as to how slow X11 is.

5. Conclusion

In summary, we have presented a method to support the ongoing research based on legacy codebases using a Virtual Time Machine. VTMs have the potential to improve the graduate student morale and productivity, and reduce project cost.

References

- [1] Chrome browser, Mar. 2013. URL <http://www.google.com/chrome>.
- [2] X.org foundation, Mar. 2013. URL <http://www.x.org/wiki/>.
- [3] P. Chapman, D. Katz, and S. Muller. Hilarious consequences of temporally distributed computing. In *Proceedings of the 8th Conference of the ACH Special Interest Group on Harry Q. Bovik*, Pittsburgh, PA, USA, 2014.
- [4] G. E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8), 1965.

- [5] G. E. Moore, VI. Cramming more components onto integrated circuits totally worked. Electronics Magazine, 238(3), 2165.
- [6] S. Seshan. In Leaked Proceedings of the CSD Black Friday Meetings, Pittsburgh, PA, USA, 2012. CMU Computer Science Department.



SIGBOVIK 2013 Paper Review

Paper 4: A Proposal for Overhead-Free Dependency Management with Temporally Distributed Virtualization

Ben Blum, Light Cone Sedentarian

Rating: REDACTED DUE TO CAUSALITY VIOLATION

Confidence: 2/4

The authors seem to be uncertain about a the work's implications to complexity theory (and don't even mention the obvious application to perplexity theory). This reviewer wonders why, if the authors have access to time travel, the paper was not more polished before submission ("before", of course, referring to the paper's causal reference frame rather than the reviewer's).

A number of important citations are missing. *Developing an Algorithm for "Enhance" Functionality in Image Processing* in SIGBOVIK 2010 first pioneered the technique of Chronological Peregrination, and *Causality 2.0: Now with Eventual Consistency* (by the Big Man Bovik himself) in SIGBOVIK 2035 will have generalized the technique to arbitrary computation.

However in my estimation the major weakness of this paper is that, by its very nature, the work cannot possibly be novel. The authors must resolve this paradox before the work can be considered complete.

The n -People k -Bikes Problem

Lancer Ångström, Dept. for Biologically-Impelled Konveyance and Excursion Research Studies

We present and solve the n -People k -Bikes Problem. This problem concerns a group of n people who possess among them k bikes, with $k < n$, and wish to travel to a destination d in a way that minimizes the time it takes the last member among them to arrive. We show that n people can, indeed, use k bikes to accelerate their travel beyond walking speed, and give an algorithm for planning bicycle allocation for general n and k .

The n -People k -Bikes Problem has far-reaching implications in many fields, including comestibility theory [3], deep-space navigation [2], maximum-jerk motion planning [4], and pop-math brainteasers.



Figure 1: *Not like this.*

1. Introduction

You're hungry. Your $n - 1$ friends, some of whom are cyclists, are also hungry. Collectively, you decide to venture out from the comfort of your respective offices on campus, through the perilously cold (and often wet) outdoors of the Pittsburgh winter, to your favorite dining establishment d .¹ During the journey, however, the advancing hunger and tormenting weather overcome your group's desire to amble idly at the leisurely walking speed of w . You glance over at your cyclist friends, who are pushing along their k bikes beside themselves, and wonder: "Might there be a way to exploit the bikes so that all n of us can get to d sooner?"

Such is the n -People k -Bikes Problem.

2. Problem Statement

The precise formulation and constraints of the problem are as follows.

¹We have only experimentally verified our research in the specific case of $d = \text{Chipotle}$, though we expect the results to generalize to all d .

Given n people and k bikes, $n < k$, all colocated, where:

1. each person can travel on foot, at walking speed w , or bike at speed b , with $w < b$,
2. mounting and dismounting a bike takes a negligible amount of time ϵ ,
3. at most one person can ride a given bike at a time (see Figures 1 and 2), and
4. bikes are stationary when not ridden,

show how all n people can arrive at a destination d in some time t less than that afforded by walking speed w .²

We now insert a page break so you can figure it out on your own if you want before moving on.

²So " k people bike the entire distance, and wait while $n - k$ people walk" is not a solution. That would be a different kind of "maximum-jerk" motion planning, which has already been thoroughly investigated in prior work [1].



Figure 2: *Stripe-Boy demonstrates correct bicycle allocation protocol. (Helmets are apparently beyond the scope of his work.)*

3. Solution

The key insight to solving the problem is that bikes can be left unattended for a time, until a walker catches up to its location. In fact in any optimal solution every bike must idle at some point.

To appreciate the method by which n people can ride k bikes to arrive with average speed greater than walking speed w , it is best to first consider the minimum nontrivial instantiation of the problem.

3.1. Specific Case: $n, k = 2, 1$

Figure 3 shows the solution for 2 people sharing 1 bike. The frames show progress after 0, 1, 1.5, 2, and 3 units of time have elapsed, assuming $w = 0.5$ and $b = 1$ (i.e., biking is twice as fast as walking). Note that the bike is idle between times 1 and 2 – for a full third of the transit time! – yet the stick figures arrive in 3 time units, a 25% improvement over walking speed, and only 50% slower than both participants having a bike of their own.

Formally we say that the *bicycle allocation* for this case is that person 0 (the blue one, if you’re reading this electronically or printed in color) rides bike 0 for distances between 0 and 1, and person 1 (the red one, if same) rides it for distances between 1 and 2. For short, we write:

$$\begin{aligned}
 p_0 &= b_0 @ 0 \\
 p_1 &= b_0 @ 1
 \end{aligned}$$

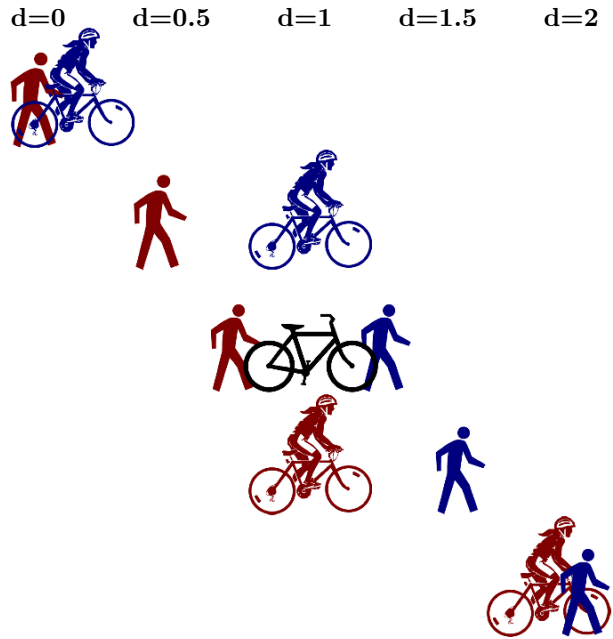


Figure 3: *The solution for $n, k = 2, 1$.*

3.2. Generalized It

Without loss of generality, let us say that d is n units of distance away (i.e., the same as the number of people). First, some observations about the problem:

1. A total of n^2 distances will be traveled. A total of nk distances will be biked. It is pointless for bikes to go backwards.
2. By symmetry, in an optimal solution, each person will bike k distance and walk $n - k$ distance. This also means that fractional distances (less than $1/n$ of the total distance) need not be considered as places to mount/dismount.
3. No more than k people can ride for any given distance unit $[i, i + 1]$. If any fewer than k people rode this way, the bikes would be underutilised. This allows us to check both validity and optimality.
4. The optimal (minimum) total travel time is $k/b + (n - k)/w$. The optimal (maximum) average speed is $nbw / (kw + bn - bk)$.

While Figure 3 plots the participants’ positions at (time,distance) coordinates, for the general instance of the problem it is more useful to plot *transportation mode* (i.e., whether a person walks or rides bike i) at (person,distance) coordinates. For example, in Figure 4 we show $n \times n$ grids for the 2, 1; 3, 2; and 5, 3 instances of the problem.

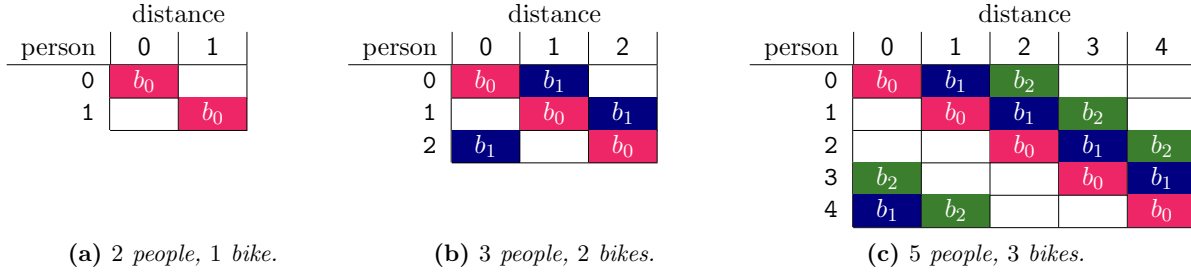


Figure 4: Bicycle allocation policies for selected instances of the n -People k -Bikes Problem. White cells represent walking; cells of different colours represent different bikes. Can you spot the French flag?

The pattern is now evident: Person i rides a bike between distance i and distance $i + k \bmod n$. To express this in the notation from the previous section, we can say: $\forall i, j$ with $0 \leq i < n$ and $0 \leq j < k$:

$$p_i = b_j @ (i + j \bmod n)$$

We now show that this strategy is both possible and optimal.

Theorem 1 (Optimality). *The n -People k -Bikes Problem is solved optimally when p_i rides between distance i and distance $i + k \bmod n$.*

Proof. By condition (3) above, it suffices to show that, at each distance unit, exactly k people ride. A simple induction on k will do nicely. \square

3.3. Minimizing Mounts/Dismounts

The astute reader will note that, while the allocations in the above tables successfully minimize total travel time, they *maximize* the number of times each rider must mount and dismount a bicycle. If we let ϵ (the time to mount or dismount) be non-negligible, we see that the travel time achieved above is actually: $k/b + (n - k)/w + 2k\epsilon$.

We might instead attempt to allocate bicycles contiguously, like in Figure 5 below. Here, $n - k + 1$ riders can ride “monogamously”, though $k - 1$ people will have two riding sessions with a walking intermission. This gives us a better travel time of

		distance				
person	0	1	2	3	4	
0	b_0	b_0	b_0			
1		b_1	b_1	b_1		
2			b_2	b_2	b_2	
3	b_1			b_0	b_0	
4	b_2	b_2			b_1	

Figure 5: Each person mounts/dismounts 2 or 4 times.

$k/b + (n - k)/w + 4\epsilon$, which is a strict improvement whenever $k > 2$.

Theorem 2 (Contiguous Allocation). *For all n and k , an optimal bicycle allocation exists in which no rider mounts and dismounts more than four times, and is given as follows:*

$$p_i = b_{(i \bmod k)} @ \{i \rightsquigarrow i + k\} \quad i \leq n - k$$

$$p_i = \begin{matrix} b_{(i \bmod k)} @ \{i \rightsquigarrow n\}, \\ b_{(i+k-n)} @ \{0 \rightsquigarrow i + k - n\} \end{matrix} \quad i > n - k$$

where $b_x @ \{y \rightsquigarrow z\}$ expands to $b_x @ (y)$, $b_x @ (y + 1)$, \dots , $b_x @ (z - 1)$.

Proof. By induction on k . \square

4. Evaluation

In Figure 6 we find our visualizings. The horizontal line on top represents the previous state of the art, in which some participants walk the entire distance. The bottom line represents the ideal case, in which there are enough bikes for everyone to ride together.

Here we have plotted the travel time for various values of n, k against different configurations of the b/w ratio; i.e., how much faster it is to bike than to walk. We hope this presentation will be useful in many different terrains; for example, if the destination is downhill, b/w may be very high indeed.

5. Conclusion

You and your friends have arrived at d at the unprecedentedly fast speed of $nbw/(kw + bn - bk)$, and are warming up indoors and enjoying dinner. “Wow,” you say, “I sure am glad everyone thought to make n copies of their bike-lock keys!”

Time Savings in the n-People k-Bikes Problem

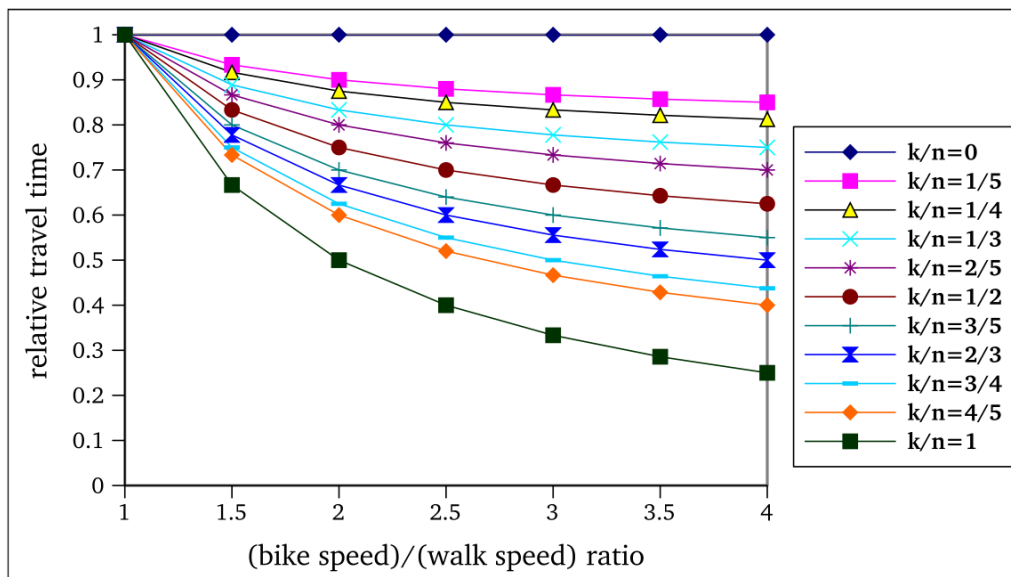


Figure 6: Speedup achieved in the n-People k-Bikes Problem. As always, lower is better.

References

- [1] F. Anchovie*. Maximum-jerk motion planning. In *Proceedings of the 2st Annual Intercalary Workshop about Symposium on Robot Dance Party in Celebration of Harry Q. Bovik's 2⁶th birthday*, 2008.
- [2] N. Beckman. Arkanoid: *Breaking Out* of a finite space. In *Proceedings of the 3th Annual Intercalary Workshop about Symposium on Robot Dance Party in Celebration of Harry Q. Bovik's 2⁶th birthday*, 2009.
- [3] Carlo Angiuli (ed.). Comestibility theory. Track 2 of the 6nd Annual Intercalary Workshop about Symposium on Robot Dance Party in Celebration of Harry Q. Bovik's 2⁶th birthday, 2012.
- [4] J. Kua and P. Velagapudi. Optimal jerk trajectories. In *Proceedings of the 2rd Annual Intercalary Workshop about Symposium on Robot Dance Party in Celebration of Harry Q. Bovik's 2⁶th birthday*, 2008.

A. Scratch Work

These are some more tables we drew when deciding whether Theorem 2 was even true at all, and they were too colorful not to include.

	distance						
person	0	1	2	3	4	5	6
0	b_0	b_0	b_0				
1		b_1	b_1	b_1			
2			b_2	b_2	b_2		
3				b_0	b_0	b_0	
4					b_1	b_1	b_1
5	b_1					b_2	b_2
6	b_2	b_2					b_0

	distance						
person	0	1	2	3	4	5	6
0	b_0	b_0	b_0	b_0	b_0		
1		b_1	b_1	b_1	b_1	b_1	
2			b_2	b_2	b_2	b_2	b_2
3	b_1			b_3	b_3	b_3	b_3
4	b_2	b_2			b_4	b_4	b_4
5	b_3	b_3	b_3			b_0	b_0
6	b_4	b_4	b_4	b_4			b_1

Figure 7: To those who printed in black-and-white and find these illegible, we say: stop reinforcing the color binary!



SIGBOVIK 2013 Paper Review

Paper 13: The n -People k -Bikes Problem

James McCann, TCHOW

Rating: A+++++++ strong work, would read again

Confidence: $\infty/4$, even though I only skimmed the second half of the paper

Finally, a practical result in computer science.

Irate Driver, The Road

Rating: Requires Revision

Confidence: 4/4

Look, I'm not a terrible person, I'm fine sharing the road with y'all cyclists. Sure, you sometimes move slower than traffic, but mostly you keep pace, and I don't mind it. (Though it can get a bit scary when you blend in with the surroundings – I really don't want to hit you folks.)

But leaving your bike just sitting in the middle of the street and continuing on foot? Seriously?

I mean, in the last week alone I've had to carefully drive around at least six bikes just sitting in the middle of the street between CMU campus and $d = \text{Chipotle}$. At first, I suspected this was a sign of the rapture, with the virtuous cyclists being taken off while slovenly drivers were left to experience the tumultuous end times. And then I found a preprint of this paper near one of the seemingly abandoned bikes.

Please, for the love of order in chaos, for the light of reason in the darkness of ignorance, for the sake of my commute: *include a notice that leaving your bike in the street is **NON-OPTIMAL**.*

Cyclolector, The Cycle-Cave

Rating: Resigned Acceptance

Confidence: 4/4

The correct solution is clearly $n - k$ more bikes.

The Problem of Heads of a Fighting Force from Long Ago

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI In Many Lands

Sometimes, in a group of computers, there will be a broken part that will tell confusing facts to the rest of the computers in the group. If we want to be able to trust the entire group, it must be able to ignore these confusing facts when deciding what to do. We can talk about this situation by playing make-believe about the heads of a fighting force from long ago, who are waiting with their people around a bad-guy city. The people must agree upon a shared fighting plan, but can only talk to each other by sending a person who carries a letter with orders written on it. However, one or more of the heads may be bad guys who will try to confuse the others. The problem is to find a way for the good guys to talk to each other, without knowing who the bad guys are, to make sure they can agree on what to do anyway. We will show that, if the good guys use only word-of-mouth, this problem is possible if and only if more than two-thirds of the heads are good guys. This means that a single bad guy could confuse two good guys. If the good guys use signed written letters, which means a bad guy couldn't pretend their letter came from someone else, the problem is possible for any number of heads and possible bad guys.

Words About What This Paper is About: C.2.4. [**Computer-Talking Groups**]: Groups of many computers—*groups of brains of computers*; D.4.4 [**Computer Brains**]: Managing Talking—*talking in groups*; D.4.5 [**Computer Brains**]: Being Trusted—*accepting faults*

Big Picture Words: Plans, Being Trusted

More Key Words and Groups of Words: Friends agreeing with themselves

1. OPENING WORDS

If you have a group of computers that you want to be able to trust, it must be able to live even if one of its parts breaks. A broken part may act in a way that people often don't think about – that is, sending confusing facts to different parts of the computer-group. We will talk about this kind of breaking-problem, using make-believe, as the Problem of Heads of a Fighting Force from Long Ago.

We imagine that several parts of the fighting force from long ago are waiting outside a bad-guy city, each group controlled by its own head person. The heads can talk with one another only by sending a person who carries a letter with the orders written on it. After watching the bad guys, they must decide upon a shared plan of attack. However, some of the heads may be bad guys, trying to stop the good guys heads from agreeing. The heads must have an plan to make sure that:

A. All good guys decide upon the same plan of how to act.

This work was helped in part by the People Who Go To Space under plan NAS1-15428 Change 3, the Group for Being Safe Against Flying Fire Guns under plan DASG60-78-C-0046, and the Fighting Force Work Office under plan DAAG29-79-C-0102. Where the people who wrote this live: Computer Study Work Place, SRI In Many Parts of the World, 333 Flying Animal's Wood Street, Park with a Person's Name, CA 94025.

You are allowed to take all or part of this paper without paying as long as the you don't make or give papers for making money, and the notice that Agreeing-group for Computer Making owns this paper, and the name of the paper and its date appear, and you give notice that if anyone takes it, it's because the Agreeing-group for Computer Making allowed them. If you want to take this paper in another way, or to put it in a book or on computers where people can see it, you need to pay and/or to ask us to let you.

©1982 ACM 0164-0925/82/0700-0382 \$00.75

ACM Talking about Computer Word-Sets and Groups, Part 4, No. 3, Month Seven 1982, Pieces of paper 382-401.

The good guy heads will all do what the plan says they should, but the bad guy heads may do anything they wish. The plan must make sure that situation A happens no matter what the bad guys do.

The good guys should not only agree, but should agree upon a plan that makes sense. So, we also want to make sure that

B. A small number of bad guy heads can't cause the good guy heads to take a bad plan.

It is hard to make clear what we mean in Situation B, since it needs you to say exactly what a bad plan is or isn't, and we do not attempt to do so. Instead, we consider how the heads reach a way to agree. Each head watches the bad-guy city and tells what he or she sees to the others. Let $\text{fact}(i)$ be the facts told by the head with number i . Each head uses some way to put together the facts $\text{fact}(1) \dots \text{fact}(n)$ into a single plan of what to do, where n is the number of heads. We can make Situation A happen by having all heads use the same way for putting together the facts, and make Situation B happen by using a way that can be trusted. Think about this case: if the only thing to agree on is whether to attack or run away, then $\text{fact}(i)$ can be Head i 's thought of which way is best, and the real plan can come from which way has more heads that want to do it. A small number of bad guys can change the way only if there were almost the same number of good guys who wanted to do each way, in which case not either way could be called bad.

While this approach may not be the only way to make situations A and B happen, it is the only one we know of. It needs a way for the heads to tell their facts $\text{fact}(i)$ to one another. The most clear way is for the head number i to send $\text{fact}(i)$ by a person who carries a letter to each other head. However, this does not work, because making situation A happen needs every good guy head to read the same facts $\text{fact}(1) \dots \text{fact}(n)$, and a bad guy head may send different facts to different heads. For situation A to happen, the following must be true:

1. Every good guy head must hear the same facts $\text{fact}(1) \dots \text{fact}(n)$.

Situation 1 means that a head can't in all cases use a fact of $\text{fact}(i)$ read straight from the head number i , since a bad guy head number i may send different facts to different heads. This means that without care, in meeting situation 1 we might make it possible for the heads to use a fact for $\text{fact}(i)$ different from the one sent by head number i —even though that head is good. We must not allow this to happen if situation B is to be met. Think about this case: we can't allow a few bad guys to cause the good guys to act as if the facts were “run away”, . . . , “run away” if every good guy said “attack”. So, we need the following thing for each i :

2. If head number i is good, then the fact that he or she sends must be used by every good guy head as the fact for $\text{fact}(i)$.

We can say situation 1 another way by saying that for every i (whether or not the head number i is a good guy),

1'. Any two good guy heads use the same meaning of $\text{fact}(i)$.

Situations 1' and 2 both use just the single fact that was sent by head number i . Because of this, we can think about the smaller problem of how a single head sends their fact to the others. We'll talk about this by talking about a controlling head sending a letter to her friends, causing the following

problem.

The Problem of Heads of a Fighting Force from Long Ago. A controlling head must send an order to her $n-1$ friends such that

FA1. All good-guy friends listen to the same order.

FA2. If the controlling head is a good guy, then every good guy friend listens to the order she sends.

These situations, FA1 and FA2, are called the *friends agreeing* situations. Note that if the controlling head is a good guy, then FA1 follows from FA2. However, the controlling head need not be good.

To fix our first problem, head number i sends their fact of $\text{fact}(i)$ by using an answer to the Fighting Force Heads Problem to send the order “use $\text{fact}(i)$ as my fact”, with the other heads acting as the helping friends.

2. SHOWING WHAT ISN'T POSSIBLE

The Problem of Heads of a Fighting Force from Long Ago seems simple, but it is actually not. It is hard because of the surprising fact that if the heads can send only talk to each other out loud, then any plan can only work if more than two-thirds of the heads are good guys. Now, when we say “talking out loud”, we mean that the words are completely under the control of the person saying them, so a bad guy could say anything he or she wanted to, even “Your good-guy friend said to do such-and-such!” This sort of speaking is the same as how computers usually speak to one another. In Part 4, we consider signed, written letters, for which this is not true.

We now show that with spoken words, no plan for three people can handle a single bad guy. To keep things simple, we consider the case in which the only things to do are “attack” or “run away”. Let us first think about the situation pictured in Figure 1 in which the controlling head is a good guy, and sends an “attack” order, but the second friend is a bad guy and tells the first friend that he heard a “run away” order. For FA2 to happen, the first friend must listen to the order to attack.

Now consider another situation, shown in Figure 2, in which the controlling head is actually a bad guy, and sends an “attack” order to the first friend and a “run away” order to the second friend. The first friend does not know who the bad guy is, and she can't tell what the controlling head actually said to the second head. So, the situations in these two pictures appear exactly the same to the first friend. If the bad guy lies all the time, then there is no way for the first friend to know which situation is happening, so she must listen to the “attack” order in both of them. So, any time the first friend hears an “attack” order from the controlling head, she must listen to it.

However, in the same way we could show that if the second friend hears a “run away” order from the controlling head then he must listen to it even if the first friend tells him that the head said “attack”. Because of this, in Figure 2, the second friend must listen to the “run away” order while the first friend listens to the “attack” order, which means situation FA1 doesn't happen. So, there is no way for three people to agree if one of them is a bad guy.

This way of thinking about it may appear right, but you should think hard before believing such hand-waving reasoning. Although what we said is right after all, we have seen other “ways of thinking about it” that are totally wrong. We know of no area in the study of computers or the study of numbers in which hand-waving reasoning could more easily lead to wrong answers than in the study of this type of problem. For a stronger way of thinking about why this isn't possible, you should read [3]. Using this answer, we can show that no plan can work for a given number of heads if one third of them are bad guys.

You might think that it's so hard to fix the Fighting Force Heads Problem because the heads need to agree with each other completely. Actually, this is not the case, and only-sort-of agreeing with each

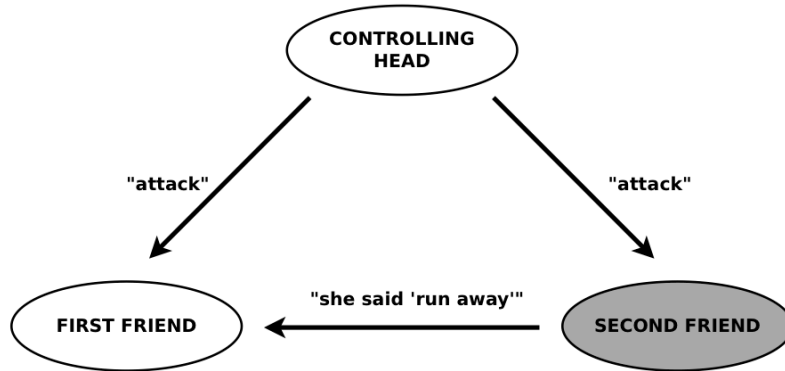


Fig. 1. The second friend is a bad guy.

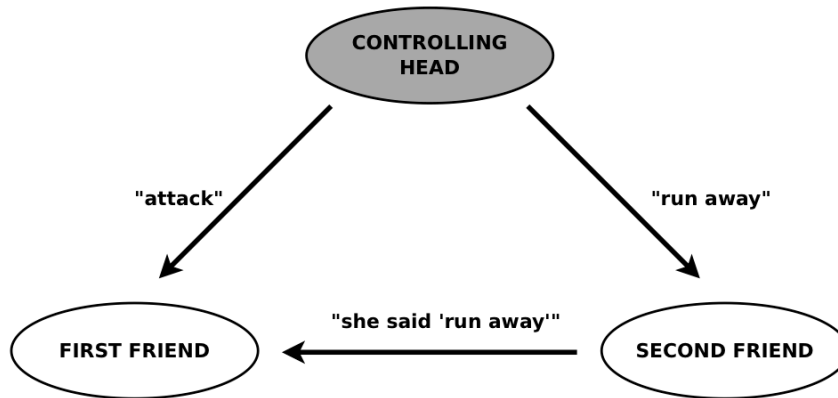


Fig. 2. The controlling head is a bad guy.

other is just as hard. Suppose that instead of trying to agree on a complete plan to attack, the heads must agree only upon a set of times, during which the attack should happen. We will say the controlling head needs to order the time of the attack, and the following two situations need to happen:

FA1'. All good guys attack within 10 minutes of one another.

FA2'. If the controlling head is a good guy, then every other good guy attacks within 10 minutes of the time given in the controlling head's order.

(We are supposing that the orders are given and thought about the day before the attack and that the time at which an order is heard doesn't matter—only the attack time given in the order matters.)

Like the Problem of Heads of a Fighting Force from Long Ago, there is no answer to this problem except when more than two-thirds of the heads are good guys. We make sure this is true by first showing that if there were an answer for three people that were okay with one bad guy, then we could answer the first problem, which we already showed was not possible. Suppose the controlling head wishes to send an “attack” or “run away” order. We'll say that she can show she wants to attack by

ordering an attack time of 1:00, or he could show he wants to run away by ordering an attack time of 2:00. Each of her friends, that is, the other heads, will listen to her order in the following way.

- (1) After hearing the attack time, a head does one of the following:
 - (a) If the time is 1:10 or earlier, then attack.
 - (b) If the time is 1:50 or later, then run away.
 - (c) Or else, continue to step (2).
- (2) Ask the other not-controlling head what they did in step (1).
 - (a) If they did either of the first two things, then do the same thing they did.
 - (b) Or else, run away.

It follows from FA2' that if the controlling head is a good guy, then her good guy friends will hear the right order in step (1), so situation FA2 happens. Also, FA1 follows from FA2, so we only need to make sure of FA1 when we suppose the controlling head is a bad guy. Since there is at most one bad guy, this means that both of her friends are good guys. It follows from FA1' that if one friend decides to attack in step (1), then the other can't decide to run away in step (1). So, either they will both do the same thing in step (1) or at least one of them will wait until step (2). In this case, it is easy to see that they both decide to do the same thing, so FA1 happens. This means we just built a three-person answer for the first problem, which is not possible. So, we can't have a plan for three people that makes FA1' and FA2' happen if one of them is a bad guy.

Notice how we had one of the heads pretend to be m other heads at the same time. We can now do that again to make sure that no answer with three times m (or fewer) people can ever be okay with m bad guys. The way of making sure that's true is like the one for the first problem, and is left for the person reading this paper to do on their own.

3. AN ANSWER FOR WHEN THE HEADS USE SPOKEN WORDS

We showed above that for an answer to the Problem of Heads of a Fighting Force from Long Ago using spoken words to be okay with m bad guys, there must be more than three times m heads in total. We will now give an answer that works for more than three times m heads. However, we first want to make clear exactly what we mean by "spoken words". Each head is supposed to follow some plan that tells them to send some facts to the other heads, and we suppose that a good guy will actually follow the plan. When we say "spoken words", we mean that we're supposing the following things about the way the heads talk to each other:

- A1. Every word that is said can be heard by the person who is meant to hear it.
- A2. Any person who hears something knows who said it.
- A3. If someone tried to say something, but their words went missing, you can know that that happened.

By supposing A1 and A2, we make sure a bad guy can't cause two other people to not be able to talk each other or to think they said things that they didn't. By supposing A3, we make sure a bad guy can't make people not be able to decide something by not saying anything at all.

The answers that we give in this part of the paper and in the following one only work if each head is able to speak to each other head without needing anyone in between to help. We would talk about answers which do not need this to happen, but we are leaving that part out of our paper so the paper does not get too long.

A bad-guy controlling-head may decide not to send any order. Since the other heads must decide to follow some order, they need something to do if no-one tells them to do anything. We'll say RUN AWAY will be this order.

We will show how to run a Spoken Words plan by which a controlling-head sends an order to the rest of the heads. We'll talk about this plan by a way of supposing that, as long as the plan works for some number of heads, it can also work for one more than that many heads. So if we can show that, then we know it works for all possible numbers of heads.

We show that the Spoken Words plan, with some number m , is an answer to the Problem of Heads of a Fighting Force from Long Ago for more than three times m many heads, as long as there are m or fewer bad guys.

The Spoken Words plan where m is nothing at all (which means there are no bad guys):

- (1) The controlling head says what she decided to do to every other head.
- (2) Each other head does the thing they heard the controlling head say, or does RUN AWAY if they didn't hear anything.

The Spoken Words plan where m is one more than some other number j :

- (1) The controlling head says what she decided to do to every other head.
- (2) For each number i , let fact_i be the fact that head number i heard from the controlling head, or else be RUN AWAY if they hear nothing. Head number i acts as the controlling-head in the Spoken Word plan for one fewer than m , to say fact_i to the other heads.
- (3) For each number i , and each number j not the same as i , let $\text{fact}_{i,j}$ be the fact that head number i heard from head number j in step (2) (using the Spoken Word plan for one fewer than m), or else RUN AWAY if they heard nothing. Head number i will do the thing that they heard most often, from all the $\text{fact}_{i,j}$.

To understand how this plan works, we consider the case where m is one and there are four total heads. Figure 3 shows what the second head hears when the controlling head says some fact f , and the third head is a bad guy. The third head says some other fact g . In step 3, the second head then heard f two times and g once, so decides to do the right thing, which is f .

Next, we see what happens if the controlling head is a bad guy. Figure 4 shows what the other heads hear if the controlling head says whatever she wants, which we'll say are x , y , and z , because they can all be different. Each other head hears all three different orders, so they all decide to do the same thing in step (3), no matter if x , y , and z are the same.

The bigger Spoken Word plan, where there are two or more bad guys, uses the smaller plan many different times. This means that for more than one bad guy, each head will have to talk to the other heads many different times. But there has to be a way for each head to tell which facts are which. You can make sure there is a way to do this by having each head, just before saying some fact fact_i , also says the number i , in step (2).

To make sure the Spoken Words plan is right for any number of bad guys, we first make sure the following expression is true.

EXPRESSION 1. *For any numbers m and k , the Spoken Words plan makes situation FA2 happen if there are more than $2k + m$ heads and at most k bad guys.*

REASON THAT IS TRUE. We can show the reason this is true by thinking about it in a way of supposing it is true for a given number then showing it must also be true for one more than that number. In this case the number we will pay attention to is m . FA2 only talks about what must happen if the controlling head is a good guy. First, using A1, it is easy to see that the most simple plan (the Spoken Words plan for no bad guys at all), so the expression is true for the starting case. We now suppose the expression is true for one fewer than some number m , and show it is true for m .

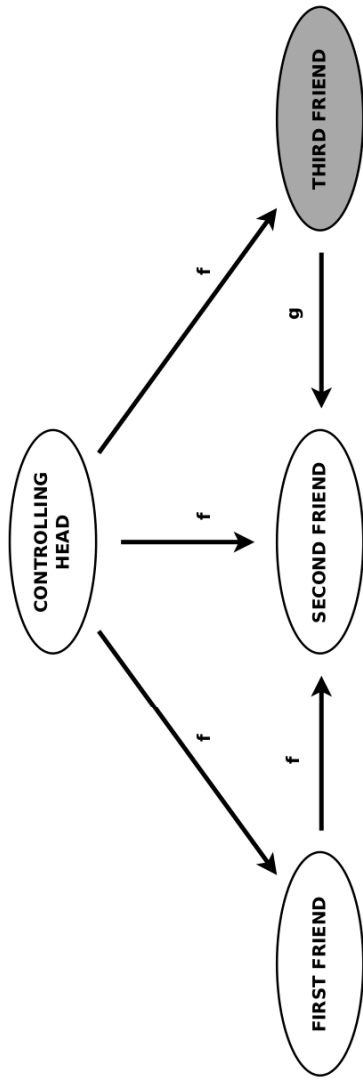


Fig. 3. Spoken Words plan, where the third head is a bad guy.

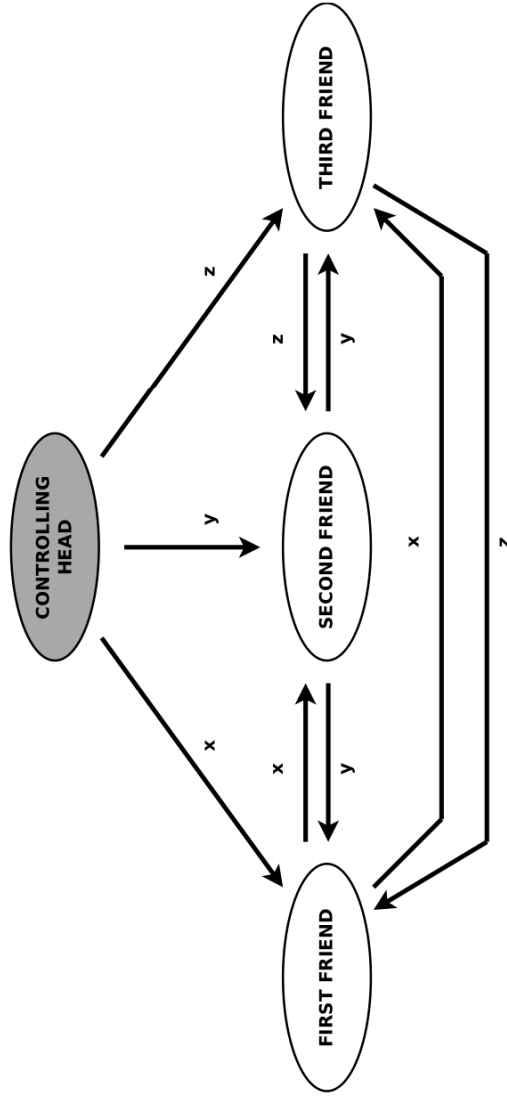


Fig. 4. Spoken Words plan, where the controlling head is a bad guy.

In step (1), the good-guy controlling head sends an order to all $n - 1$ of their friends. In step (2), each good-guy friend runs the Spoken Words plan for $m - 1$ bad guys with $n - 1$ heads. Since we supposed n is bigger than $2k + m$, we know $n - 1$ is bigger than $2k + (m - 1)$, so we can use the thing we supposed to show that every good-guy friend decides to do what the controlling head told them. Since there are at most k bad guys, and $n - 1$ is bigger than $2k + (m - 1)$, which in turn is bigger than two times k , then more than half of the friend-heads are good guys. So, each good-guy friend decides to do what the controlling head told them in the third step, which makes sure that FA2 happens. \square

The following expression says that the Spoken Words plan is a right answer to the Problem of Heads of a Fighting Force from Long Ago.

EXPRESSION 2. For any number of bad guys, the Spoken Words plan makes sure both Friends-Agreeing situations happen if there are more than three times as many people as bad guys.

REASON THAT IS TRUE. We show the reason this is true by the same way of supposing as before. If there are no bad guys, then it is easy to see that the easiest Spoken Words plan makes situations FA1 and FA2 happen. So, we suppose that the expression is true for the Spoken Words plan for $m - 1$ bad guys, and show it is true for m bad guys.

We first consider the case in which the controlling head is good. By letting k be the same as m in Expression 1, we see that the Spoken Words plan for m bad guys makes FA2 happen. FA1 follows from FA2 if the controlling head is good, so we need only make sure FA1 happens in the case that the controlling head is a bad guy.

There are at most m bad guys, and the controlling head is one of them, so at most one fewer than m of their friends are also bad guys. Since there are more than three times m people in total, there are more than $3m - 1$ friends, and one fewer than three times m is bigger than three times one fewer than m . So we can use the thing we supposed to show that the Spoken Words plan for $m - 1$ bad guys makes FA1 and FA2 happen. Because of that, any two good-guy friend-heads decide to do the same thing in the third step of the plan, which makes FA1 happen. \square

4. A PLAN FOR WHEN BAD GUYS CAN'T PRETEND THEIR WORDS CAME FROM SOMEONE ELSE

As we saw from the situations in Figures 1 and 2, it is the fact that bad guys can lie that makes the Fighting Force Heads Problem so hard. The problem becomes easier to find an answer for if we can make sure they don't lie. One way to do this is to let the heads send signed letters to each other. Using computers, there are ways to sign your letters where you use some very big numbers that make it very hard for a bad guy to pretend a signed letter was written by someone else or says something different. We'll suppose the heads use this way of signing letters. So, already supposing A1, A2, and A3, we will also suppose the following:

- A4 (a) A bad guy can't write a signed letter and pretend that a good guy wrote it, and also can't change a signed letter that was already written to say something else.
 (b) Anyone can make sure that a signed letter came from the person who signed it.

Note that we don't suppose anything about letters that a bad guy signed. One important thing this means is that bad guys can write signed letters and pretend they came from other bad guys. So we are even letting bad guys work together instead of acting alone.

Now that we can write signed letters, the thing we said earlier about needing four or more heads in order to be okay with one bad guy is no longer true. In fact, there is an answer for three heads now. We now give an answer where any number of heads are okay with any number of bad guys (though the problem doesn't make any sense if there are fewer than two good guys).

Usually, when signing a letter, you write your name at the bottom of the letter. But since we are using computers and very big numbers for this stronger sort of signing, the thing you write after the letter is numbers, instead of your name. We will call these “signing-numbers”.

In our answer, the controlling head sends a signed order to each of her friends. Each friend then adds his or her signing-numbers to that order and sends it to the other friends, who add their signing-numbers to that order and send it to everyone else, and so on. This means that each head must be given one signed letter and sign it and send it to several other people.

In the following answer, when we say $f : i$, we mean the fact f signed by head number i . So, $f : j : i$ means the fact f signed by head j , and then that fact-and-signing-numbers signed again by head i . We will say Head One is the controlling-head, who gives the orders. In this plan, each head remembers a set V_i which is made of all (not pretended) signed orders he or she has been given so far. (If Head One is a good guy, then this set should never have more than one thing in it.)

Do not confuse V_i , the set of orders that a head was given, with the set of words that were said to them. There may be many different words spoken about the same order.

Spoken Words Plan for m bad guys.

At the start, each V_i is empty.

- (1) The controlling head signs and sends her order to every friend head.
- (2) For each i :
 - (a) If head number i is given a letter of the form $f : 1$ from the controlling head and he has not yet heard any order, then
 - i. he lets V_i become just $\{v\}$;
 - ii. he sends the signed letter $v : 1 : i$ to every other head.
 - (b) If head number i gets a letter of the form $f : 1 : j_1 : \dots : j_k$ and v is not in the set V_i , then
 - i. he adds v to V_i ;
 - ii. if k is less than m (the number of bad guys), then he sends the letter $f : 1 : j_1 : \dots : j_k : i$ to every head besides the ones with numbers j_1 through j_k .
- (3) For each i : When head number i will get no more letters, he or she decides to follow any order in the set V_i , or RUN AWAY if the set is empty.

Note that in step (2), the heads ignore any letter that talks about an order already in the set V_i .

Figure 5 shows the Signed Letters plan for the case of three heads when the controlling head is a bad guy. The controlling head sends an “attack” order to one head and a “run away” order to another. Both friend-heads get the same two orders in step (2), so after step (2) V_2 and V_3 are both {“attack”, “run away”}, and they both decide what to do the same way. Notice that here, which is not like the situation in Figure 2, the friends know that the controlling head is a bad guy because she signed two different orders, and A4 states that only she could have made those signing-numbers.

5. CLOSING WORDS

We have presented several answers to the Problem of Heads of a Fighting Force from Long Ago, in several different situations. The plans in these answers can take a lot of time to run and can also need a lot of letters to be sent. Both the Spoken Words plan and the Signed Letters plan might need each order to be sent more than m times, where m is the number of bad guys.

In other words, each head may have to wait for orders that came from the controlling head and were then passed on by m other heads. Other people who study computers have shown that this must be true for any answer that can be okay with with m bad guys, so our answers are as good as possible.

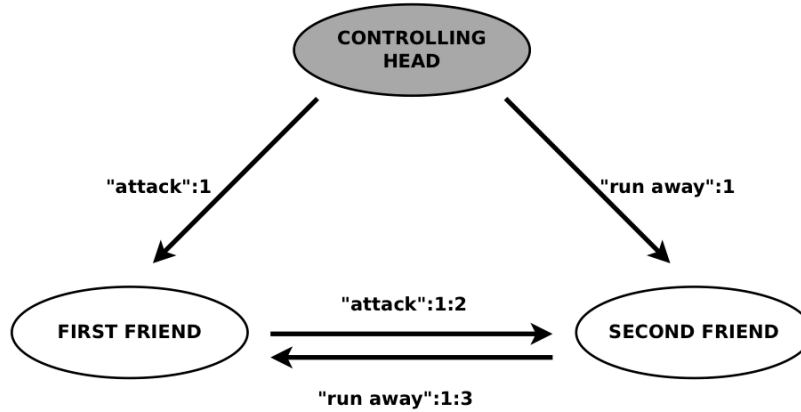


Fig. 5. Signed Letters plan, with the controlling head as a bad guy

The Spoken Words plan and the Signed Letters plan need you to send up to $(n - 1)$ times $(n - 2)$ times (and so on. . .) times $(n - m - 1)$ letters or words. You can make the number of different times you need to send lower by putting letters or words together and sending them at the same time. It may also be possible to lower how much you actually need to say at all, but we have not studied this enough to be sure. However, we expect that a large number of letters will still be needed.

Making a group of computers that you can trust even when there might be bad guys that try to confuse you in whatever way they want is a hard problem. It seems like any plan to do this will need to take a lot of time or space. The only way to make it need less is to start supposing things about the way things might go wrong. One such way is to suppose that a computer may stop responding but will never respond with something confusing. However, when you need to trust your computers very very much, you can't suppose that sort of thing, and you need to spend all time and space it takes to find an answer for the Fighting Force Heads Problem.

PAST THINGS THAT WE BUILT ON

1. MONROE, R. Up Goer Five. *XKCD Computer Funny Pictures*, <http://xkcd.com/1133/>, 2013.
2. SANDERSON, T. The Up-Goer Five Word-Writing Computer Game. <http://splasho.com/upgoer5/>, 2013.
3. PEASE, M., SHOSTAK, R., AND LAMPORT, L. Reaching a Way to Agree Even When There are Faults. *J. ACM* 27, 2 (Month Four 1980), 228-234.

This paper was given to us in Month Four 1980; accepted in Month Ten-and-One 1981; and was written again using only the ten hundred most used words in Month Four 2013. In the interest of not being too long, some parts of this paper have been left out. Sorry to the people who wrote this paper the first time.

duoludo: a game whose purpose is games

<http://dwrensha.ws/duoludo>

David Renshaw

Designing games is hard. In order to create an engaging and coherent space of interactions for players to explore, a game designer must develop a keen awareness of players' possible mindsets and intentions throughout the space. It is easy to overlook points in the space that appear to be inaccessible, and it is difficult to avoid biases about players' inclinations even at obviously accessible points. A game designer needs to be able to see things as a newcomer. Human beta testers can help in developing this awareness, but they usually are inefficient explorers. They often share many of the biases of the designer and tend to get stuck at all of the same points, especially if they all start playing from the same point. Automated testing can also help, but unless it incorporates some sort of sophisticated artificial intelligence, it usually fails to achieve adequate coverage. In any case, it gives no clue as to how human players, the intended audience, would react in corner cases. Are there any other options for a game designer in search of feedback?

Recent work has shown that the powers of human intelligence and computers can be combined and harnessed to perform tasks that neither can perform alone. Examples of this principle in action include systems for protein folding [1], image labeling [2], and text translation [3]. In each of these systems, computers perform otherwise intractable tasks by offloading to humans the parts for which humans are particularly well suited. The humans, in turn, find these subtasks so stimulating and rewarding that they are willing to perform them for free. To make participation even more enjoyable, these systems may also add game mechanics such as achievement levels and leaderboards.

Duoludo is a prototype system that aims to apply this principle to the problem of providing better feedback for game designers. Duoludo takes as input a game, which must be written in javascript and must implement a simple interface allowing duoludo to control it. Duoludo serves bite size chunks of the game to players on the Internet, recording their inputs. Because duoludo gets to choose which chunks to serve, it can steer players toward unexplored regions. Moreover, stripped of their original context, these chunks will elicit a more diverse set of responses than what would be observed in traditional beta testing. The paths through the game generated in this way may then be stitched together, replayed, and analyzed in whatever way the game designer finds most useful.

References

1. <http://fold.it>
2. <http://www.gwap.com>
3. <http://www.duolingo.com>

The First Level of Super Mario Bros. is Easy with Lexicographic Orderings and Time Travel ...after that it gets a little tricky.

Dr. Tom Murphy VII Ph.D.*

1 April 2013

Abstract

This paper presents a simple, generic method for automating the play of Nintendo Entertainment System games.

Keywords: computational super mario brothers, memory inspection, lexicographic induction, networked entertainment systems, pit-jumping, ...

1 Introduction

The Nintendo Entertainment System is probably the best video game console, citation *not* needed. Like many, I have spent thousands of hours of my life playing NES games, including several complete playthroughs of classics like Super Mario Bros., Bionic Commando, Bubble Bobble, and other favorites. By the year 2013, home computers have become many orders of magnitude faster and more capacious than the NES hardware. This suggested to me that it may be time to automate the playing of NES games, in order to save time.¹ In this paper I present a generic technique for automating the playing of NES games. The approach is practical on a single computer, and succeeds on several games, such as Super Mario Bros.. The approach is amusingly elegant and surprisingly effective, requires no detailed knowledge of the game being played, and is capable of novel and impressive gameplay (for example, bug exploitation). **Disclaimer for SIGBOVIK audience: This work is 100% real.**

On a scale from “the title starts with Toward” to “Donald Knuth has finally finished the 8th volume on the subject,” this work is a 3. The purpose of this

paper is mainly as a careful record of the current status for repeatability and further development on this important research subject. A short video version of this paper is available for those that hate reading, at <http://tom7.org/mario>, and is the more fun way to consume the results. This page also contains audiovisual material that makes this work more entertaining (for example, its output) and source code.

The basic idea is to deduce an objective function from a short recording of a player’s inputs to the game. The objective function is then used to guide search over possible inputs, using an emulator. This allows the player’s notion of progress to be generalized in order to produce novel gameplay. A design goal is that the objective function be amusingly elegant (not at all smart, fancy, or customized to the game) in order to demonstrate that the game is reducible to such a simple objective. The search needs to be game-agnostic and practical, but since the space is exponential (256^n)[7], we need to be smart here.

The objective function, the algorithm to deduce it, the search strategy, and its implementation are all interesting and will be discussed in that order. I then discuss the results of using the approach to automate several NES games. To set the stage, I begin with a description of the NES hardware and emulation of it.

1.1 The NES hardware and emulation

The NES is based around an 8-bit processor running at 1.79 MHz, the Ricoh 2A03. 8 bits is really small. You can see them all right here: 00001111. It’s no coincidence that each controller also has 8 buttons: Up, Down, Left, Right, Select, Start, B and A. It has only 2048 bytes of general purpose RAM. (There is also some special purpose RAM for graphics, which we ignore in this work.) 2048 bytes is really small. You can see them all in Figure 1. As a result, NES programs are written to use memory efficiently and straightforwardly; usually there are fixed memory locations used for all the

*Copyright © 2013 the Regents of the Wikiplia Foundation. Appears in SIGBOVIK 2013 with the reluctant sigh of the Association for Computational Heresy; *IEEEEE!* press, Verlag-Verlag volume no. 0x40-2A. CHF 0.00

¹Rather, to replace it with time spent programming.

critical game facts like the player’s health, number of lives, coordinates on the screen, and so on. For example, in Super Mario Bros., the single byte at location 0x757 contains the number of lives the player has. The location 0x75F contains the current world, and 0x760 the current level. The NES outputs 60.0988 frames per second, which we will just call 60 in this paper.

There are a number of emulators for NES. These work by simulating the NES hardware, for example with a 2048-byte array for its memory, and simulating the steps of its 2A03 processor on some ROM, and hooking a keyboard or joystick into the 8 bits of input. (There are of course many details to work out! But in essence emulation is just that.) This process is completely deterministic, so it is possible to record the sequence of inputs (the inputs can only be read once per video frame, so this sequence is 60 bytes per second) and play them back and get the same result. This also means that an input sequence can be computed in non-real time, either much slower or much faster than a NES would normally run. In this work we use the FCEUX[1] emulator, which is popular for its accuracy and advanced tools.

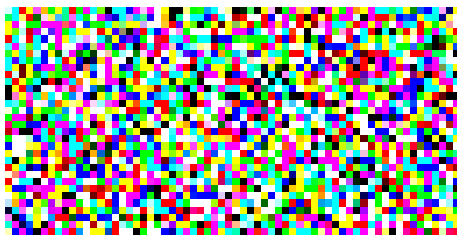


Figure 1: 2048 bytes, a 64x32 image.

2 Objective function

Bytes in memory (and sometimes 16- and 32-bit words) can contain interesting game facts like the player’s position in the level or score. The central idea of this paper is to use (only) the value of memory locations to deduce when the player is “winning”. The things that a human player perceives, like the video screen and sound effects, are completely ignored. As an additional simplification, we assume that winning always consists of a value *going up*—either the position in the level getting larger, the score getting larger, the number of lives, the world or level number getting bigger, and so on.

This is actually a little bit too naive; for example, Mario’s overall progress through the game is represented by a pair. You start in World 1-1 and the underground

level that comes next is World 1-2 (we’ll call this $w = 1$ and $\ell = 2$). But after you discover the princess is in another castle in World 1-4, the next level is 2-1.² This can’t be represented as a single byte going up (sometimes the second part ℓ goes down when we get to a new first part w), but it can be represented as a lexicographic order on the pair $\langle w, \ell \rangle$; that is, $\langle w_1, \ell_1 \rangle < \langle w_2, \ell_2 \rangle$ if $w_1 = w_2$ and $\ell_1 < \ell_2$, or if $w_1 < w_2$ no matter the values of ℓ_1 and ℓ_2 . This matches our intuitive idea and is also mathematically nice. It also generalizes multi-byte encodings of things like your score (which can be larger than 8 bits and so is often stored in 16 or 32), including both big-endian and little-endian representations.³

More importantly, it allows the combination of semantically unrelated bytes, like: $\langle \text{world, level, screen inside the world, } x \text{ position on the screen} \rangle$ or $\langle \text{world, lives, low byte of score} \rangle$. Many orderings may describe gameplay. These orderings may be temporarily violated in normal play: Although the score always goes up, Mario’s x position may temporarily decrease if he needs to navigate around some obstacle.⁴ So, to “faithfully” represent gameplay, we will generate a set of lexicographic orderings on memory locations, with the idea that they “generally go up” but not necessarily at every step. These orderings will also have weights. The next section describes how we take a sequence of player inputs and deduce the orderings.

2.1 Deriving the objective function

In order to derive an objective function, we’ll start with an abstract subroutine that finds a single lexicographic ordering nondeterministically. This function takes in an ordered list of n memories $M_1 \dots M_n$ which all have size m bytes. For example, $m = 2048$ and $n = 100$, for the memories at each of the first 100 frames of someone playing Super Mario Bros.. It produces an ordered list of unique memory locations $L_1 \dots L_k$ (where $0 \leq L_i <$

²In case you never realized this, it is time to learn that the legendary “Minus World” of -1 is not actually a negative world, but World 36-1 being incorrectly rendered because there is no glyph for the 36th digit. The trick used to get to the Minus World just happens to leave the value 36 in that memory location rather than initializing it to a useful value. The ROM does not contain data for world 36 so it just interprets garbage data as a description of the world.

³A possible additional simplification would be to just take lexicographic orderings over bits, which then generalizes to 8-bit bytes. This is probably too crazy, but just right now I am sort of feeling like maybe I should try it, though it may be the beer.

⁴Note to self: Maybe we should give a much higher score to globally preserved objectives than to locally preserved ones. But that may presuppose that the input represents a whole playthrough?

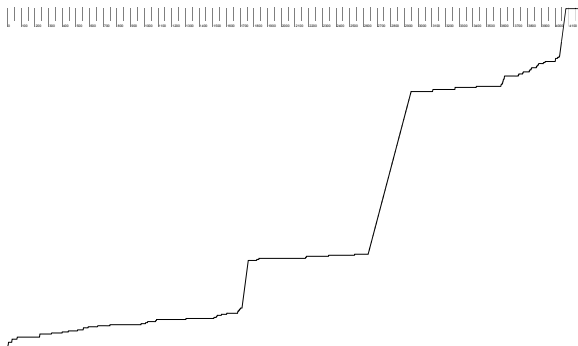


Figure 2: A single maximal tight valid lexicographic ordering for my 4,000-frame input training data to Super Mario Bros.. This function is globally nondecreasing, and is the decimal memory locations (232, 57, 73, 74, 75, 115, 130, 155, 32, 184, 280, 491, 506, 1280, 1281, 1282, 1283, 1288, 1290, 1337, 1338, 1339, 1384, 1488, 1490, 1496, 1497, 1498, 1499, 1514, 1873, 1882, 1888, 1904, 1872, 1906, 112, 113, 114, 2009, 2010, 2011, 1539). This is not a great objective function; there are long spans where all the memories are equal according to it, and the nice smooth slopes are happening during level transitions where the game is ignoring inputs (they are probably timers counting up each frame, which is why they are so smooth). Other slicing produces better objectives.

For reasons unknown—I just discovered this while generating the figure—all of the objective functions learned with this method, regardless of the nondeterministic choices, appear to have this same curve, despite using different memory locations. It may be that they are different permutations of bytes that all change simultaneously, only on the frames where there are jumps in this picture, and there are no other orderings that are tight, valid, and maximal. This is still surprising and warrants investigation.

m , that is, each is some spot in the 2048 bytes of RAM) that is a *maximal tight valid* lexicographic ordering of M . Let’s start by defining those terms just to be careful.

Given some list of memory locations $L_1 \dots L_k$ and a pair of memories M_a and M_b , we say that $M_a =_L M_b$ iff $M_a[L_1] = M_b[L_1]$ and $M_a[L_2] = M_b[L_2]$ and so on for every L_i ; that is, the bytes must be equal at each of the locations. Easy. We say that $M_a <_L M_b$ iff there exists some $p \leq k$ where $M_a[L_1] = M_b[L_1] \dots M_a[L_{p-1}] = M_b[L_{p-1}]$ and $M_a[L_p] < M_b[L_p]$. Put simply, if the two memories are not equal according to L (have the same byte at every memory location) then there is a

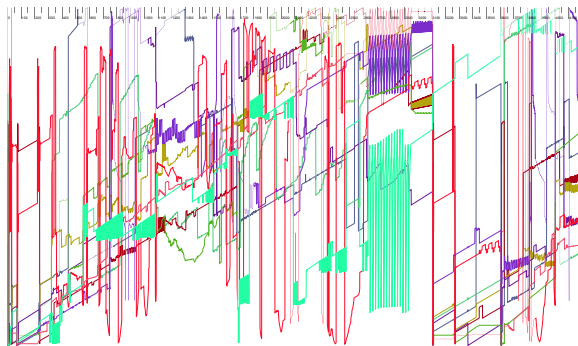


Figure 3: Ten objective functions trained on different tenths of the 4,000 inputs for Super Mario Bros.. These functions are normalized against the range of all values they take on during the movie; you can see that most are increasing locally most of the time, but many drop back to zero around the 3100th frame, when Mario reaches world 1-2. Within its 400-frame window, each objective is guaranteed to be nondecreasing.

unique first location (L_p) where they have a different byte, and that byte determines the order. $M_a >_L M_b$ is just defined as $M_b <_L M_a$; $M_a \leq_L M_b$ is just $M_a <_L M_b$ or $M_a =_L M_b$, and similarly for \geq_L , and they mean what you think so don’t worry.

Every L defines a lexicographic ordering ($<$ and $=$ operators). L is a *valid* lexicographic ordering of M if $M_i \leq_L M_{i+1}$ for $1 \leq i \leq n$; each memory is less than or equal to the next memory in the sequence. It follows that $M_i \leq_L M_j$ whenever $i < j$.

Every prefix of a valid L (including the empty prefix) is a valid lexicographic ordering as well. On a scale from useless to useful, the empty prefix is a 1 (it equates all memories), which suggests that some orderings are better than other. To give a primitive notion of “good” lexicographic orderings, we define a *maximal* valid lexicographic ordering to be L such that there are no extensions of L that are valid. An extension of $L_1 \dots L_k$ is just $L_1 \dots L_k, L_{k+1}$ (where $L_{k+1} \neq L_i$ for $1 \leq i \leq k$): Some new memory location that we put at the end of the order (in the least important position). We do not consider extending in the middle of the order or beginning, although that would make sense.

Maximal valid orderings are good and it is straightforward to produce them (a less constrained version of the algorithm below), but they have the bummer downside that memory locations that never change value for any M can be included at any position of the ordering, and all such choices are equivalent. And in fact all loca-

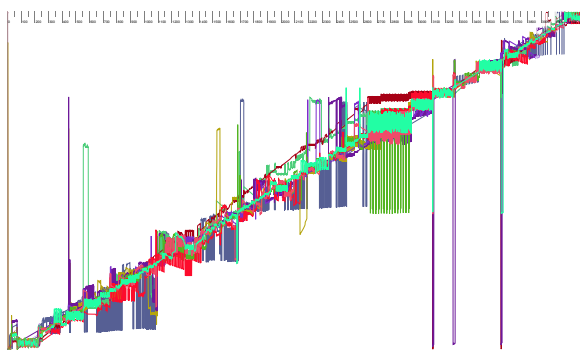


Figure 4: Ten objective functions trained on every 100th memory, starting at frame 0, frame 1, and so on up to frame 10. Normalized as usual. These objectives exhibit excellent global progress, but are locally very noisy. Among the methods I used to generate objectives, this one produces the best results (based on my intuition about what a good objective function looks like).

tions *must* be included to make the ordering *maximal*. This is bad because when M contains many locations with fixed values, we have boatloads of equivalent orderings, and they’re also longer than necessary. An *tight* valid ordering is one where for each L_i there exists at least one M_a and M_{a+1} where $M_a[L_i] < M_{a+1}[L_i]$ and $M_a[L_j] = M_{a+1}[L_j]$ for all $i < j$; that is, every location has to participate in the ordering at least once. The notion of *maximal* has to be relative to this property as well—a *tight extension* is one that produces a tight valid ordering, and a *maximal tight valid ordering* permits no tight extensions.

On a scale from simple to fancy, the algorithm to generate L from M is a 3. Given those definitions, the idea is to start with the empty prefix, which is always a tight valid lexicographic ordering but usually not maximal. We then pick a tight extension that is valid; if none exists then we are done and have a maximal tight valid ordering. Otherwise we have a new tight valid ordering, and we repeat.

The pseudocode gives a pseudoimplementation of the algorithm that is more pseudodetailed. The C++ implementation is in `objective.*`. C++ is not a good language for this kind of program but we use it because the NES emulator is written in C++.

2.2 The objective function, in practice

We can’t just use a single objective function. Choosing objective functions nondeterministically, we may get a crap one like “High byte of the score” which only goes up once during all of our memory observations. We also can’t just use all of the memory observations, because there may be brief moments that violate strict orderings, like Mario’s x coordinate temporarily decreasing to navigate around an obstacle. More starkly, the first few hundred frames of the game are almost always initialization where the memory temporarily takes on values that are not representative of later gameplay at all. In practice, we use the nondeterministic function from Section 2.1 on multiple different slices of the memory observations. We also call it many times to nondeterministically generate many different objectives. Finally, we weight the objectives by how representative we think they are.

Parameter Alert! This one of the first places where we have some arbitrary constants, which are the enemy of elegance. On a scale of ballpark to obsessively overfit, these constants are a 2; I basically looked at some graphs while developing the objective function learning part of the code to decide whether they were “good enough” and didn’t tune them after starting to observe actual performance. Some of those graphs appear in figures here. For all I know, these are really bad choices, but it was important for aesthetic reasons for the objective function to be brutish. The only reason to permit these parameters at all is that it simply does not work to have a single ordering or to use only orderings that apply to the whole memory.

Skipping. To avoid being confused by RAM initialization and menu, I ignore all memories up until the first input by the player. Including these would be especially suspicious because the RAM’s contents are not even well-defined until the program writes something to them.⁵

Slicing. I generate 50 orderings for $M_1 \dots M_n$; the whole recording starting immediately after the first

⁵Emulators tend to fix them to a specific pattern so that emulation is completely deterministic, but in real hardware they are truly uninitialized, or retain the values from the last reset or game inserted. Some semi-famous tricks involve removing and inserting game cartridges while the system is running in order to take advantage of this.

```

(* Prefix is the prefix so far (int list) and remain is the list of memory
   locations that we can still consider. Invariant is that prefix is a
   tight valid ordering on  $M$ . Returns the memory locations from remain
   that are a tight extension of the prefix. *)
fun candidates (prefix, remain) =
  let lequal = (* list of indices  $i$  where
                 $M_i =_{\text{prefix}} M_{i+1}$  *)
      let notgreater = (* members  $x$  of remain where
                         $M_i[x] > M_{i+1}[x]$  is
                        not true for any  $i$  in
                        lequal *)
          let tight = (* members  $y$  of notgreater where
                        $M_i[x] < M_{i+1}[x]$  is true
                       for some  $i$  in lequal *)
              in tight

(* Returns a maximal tight valid ordering, given a tight valid prefix and
   list of memory locations we are permitted to consider. *)
fun ordering (prefix, remain) =
  case candidates (prefix, remain) of
    (* No extensions means it's maximal. *)
    nil => prefix
  | cand =>
    let c = nondeterministically-choose-one cand
        let remain' = remove-element (remain, c)
            let prefix' = prefix @ [c]
                in ordering (prefix', remain')

```

Figure 5: Pseudocodes for nondeterministically generating a maximal tight valid lexicographic ordering on some memories M . The recursive function `ordering` just orchestrates the selection of an extension until there are no possibilities remaining, and returns it. The function `candidates` finds all the possible extensions for a prefix. First we compute `lequal`, all of the adjacent memory pairs (represented by the index of the first in the pair) where the memories are equal on the prefix. Only pairs that are equal on the prefix are interesting because these are the only ones that will even depend on the extension to the prefix when comparing the memories. We only need to consider adjacent pairs because on an a scale of exercise for the reader to proof is contained in the extended technical report, this statement is a you can figure that one out yourself. Valid extension locations are ones where the memory pairs are never increasing at that location (note that in places where pairs are not equal but strictly less on the prefix, it's perfectly fine for the extension to be greater; this is the “point” of lexicographic orderings). Finally, tight extensions are those valid ones that have at least one pair of memories where the location has a value that is strictly less.

keypress. During gameplay some values really are completely nondecreasing, like the player’s score and world/level pair. Figure 2 shows what a global ordering looks like. I also generate 3 orderings for each tenth of the memory sequence, e.g. $M_1 \dots M_{n/10}$ and $M_{n/10+1} \dots M_{2n/10}$, etc. The intention is to capture orderings that are rarely violated, or sections of the game with unusual objectives (e.g. a minigame or swimming level). Orderings generated this way look pretty random, and on a scale from solid to suspicious, I can’t vouch for them. Then I generate objectives from non-consecutive memories that are evenly spread out through the observations: Ten objectives chosen from every 100th memory, starting from the 0th frame, 1st frame, 2nd frame, and so on up to the 9th. Similarly for every 250th frame, and a single objective for memory sliced to every 1000th frame, with start positions of 0–9. The intention is to capture objectives that grow slowly over the course of a playthrough, without getting distracted by local noise.

Weighting. To reduce the importance of randomness in the learning process, and the arbitrariness of the slicing, each objective function is also assigned a weight. An ideal objective function takes on its minimal value on the first frame and maximal on the last (according to the ordering), and increases steadily throughout the observed memories. This is ideal because it allows us to just follow the gradient to reach good states. A bad objective function frequently regresses (recall that although we produce valid orderings, an ordering that is valid on some slice may not be valid for the whole sequence of memories). To weight an objective L , we first collect all of the values (the vector of values of the memory locations $L_1 \dots L_k$) it takes on throughout the observations. These may not obey the ordering. We then sort the values lexicographically and remove duplicates.⁶ Call this V . Now we can compute the *value fraction* for L on some M : Extract the vector of locations $M[L_1], M[L_2], \dots, M[L_k]$ and find the lowest index i in V where the vector is less than or equal to V_i . The value fraction VF is $i/|V|$, which is the normalized value of “how big” M is, according to L , compared to all the values we ever saw for it. The value fraction is defined and in $[0, 1)$ for all memories in the observed set.⁷ This gives us the ability to compare objectives

⁶Note to self: I’m not sure how to justify removing duplicates here. It makes $[0, 1, 1, 1, 1, 1, 10, 11]$ look the same as $[0, 1, 10, 11]$, which is probably not desirable?

⁷It is not defined when the memory is greater than all observed memories, which we will encounter later. The code returns $|V|/|V| = 1$ in that case, which is as good as anything.

on an absolute scale.⁸ Weighting an objective is now simple:

$$\sum_{i=1}^{n-1} \text{VF}(M_{i+1}) - \text{VF}(M_i)$$

We sum the differences in value functions for each consecutive pair of memories. In the case of the ideal function this is $\sum_{i=1}^{n-1} 1/n$, which approaches 1. Of course, when you think about it, this is actually the same as

$$\begin{aligned} &(\text{VF}(M_1) - \text{VF}(M_0)) + (\text{VF}(M_2) - \text{VF}(M_1)) + \\ &\quad \vdots \\ &(\text{VF}(M_{m-1}) - \text{VF}(M_{m-2})) + (\text{VF}(M_m) - \text{VF}(M_{m-1})) \end{aligned}$$

and all but $-\text{VF}(M_0)$ and $\text{VF}(M_m)$ cancel out. This means that the weight we use is just the final value minus the initial value, regardless of what happens in-between.⁹ The mean value theorem or something is probably relevant here. **Lesson about being careful:** I only realized that this code was kind of fishy when I started writing it up for SIGBOVIK. Not only did it loop over all the deltas as in the Σ expression above, but it also summed from $i = 0$ and kept track of the last value fraction at each step, thus treating the value fraction of the nonexistent memory M_0 as 0. This is wrong, because the first value fraction may be very high, which credits the objective with a positive value (e.g. 1) for that first part of the sum. Objectives that start high on the first frame are not ideal; in fact, the worst objectives start high on the first frame and steadily *decrease*. After writing this up I corrected it to the simple expression $\text{VF}(M_m) - \text{VF}(M_0)$ and the result was a huge breakthrough in the quality of the output! I had spent much time tuning the `playfun` search procedure (Section 3) and not investigated whether the objectives were being weighted properly. More on this later, but the lesson is: Bugs matter, and thinking about your code and explaining it is a good way to find bugs.

Objectives are constrained to have non-negative weights (I set the value to 0 if negative, which effectively disables it). We save the objectives and their weights to a file and then we are done with the easy part.

2.3 Motifs

The very first thing I tried with the objective function is to just do some greedy search for input sequences that increased the objective. This works terribly, because the search space for inputs is large (2^8 possibilities at each

⁸This is certainly not the only way to do it, and it has some questionable properties like ignoring the magnitude of change. But it is very simple.

⁹I think this can be improved, for example by taking the deviation from the ideal linear objective.

frame). Most are useless (it's almost impossible to press the left and right directions at the same time, and real players almost never do it); real input sequences usually do not change values 60 times per second (rather, the player holds the jump button for 10–20 frames); some button-presses and combinations are much more common than others (e.g. right+B is common for running right, but start pauses the game). Search quickly necessitates a model for inputs. Rather than do anything custom, I just use a really dumb approach: Take the observed inputs (the same ones that we learned the objective functions from) and split them into chunks of 10 inputs. Motifs are weighted by the number of times they occur in the input. There may be a single motif at the end that's fewer than 10 inputs.

Parameter Alert! Here I choose the magic number 10 for the size of input motifs. On a scale from gravitational constant to pulled it out of my ass, this is an 8. We could perhaps justify 10 as being close to the speed of input change actually possible for a human (6 button presses per second; 166ms). I believe it is possible to do much better here and the code contains a few such false starts, but using motifs was one of the biggest immediate improvements in the history of this code, so I went with it. A natural thing to try is a Markov model, although this has a free parameter too (the number of states of history). It is likely possible to do some kind of abstract interpretation where multiple different input sequences with equivalent outcomes are explored simultaneously, which might obviate the need for computing an input model from the observed play. The `playfun` algorithm below takes motifs as primitive because of the way it was developed; I'll use footnotes to describe my thinking about how to remove this.

Motifs are written to a file too and then we're done with that. This concludes the learning we do from the example input; everything else is a matter of using the objective functions and motifs to play the game.

3 Now you're playing with power

In this section I'll describe how the objective functions are used to play the game. On a scale from canonical to Star Wars Christmas Special, this algorithm is an 7. So, rather than focus on the particulars of some of the

heuristics, I'll try to give a story of the different things I tried, what motivated the ideas in the current version, and my intuitions about what is working well (or not) and why. This algorithm is called `playfun` and it can be found implemented in C++ in `playfun.cc`; some historic versions are in `playfun-*.cc`.

3.1 Basic software architecture

In order to use the emulator to search for good sequences of inputs, I needed deeper integration than just observing memory. The FCEUX emulator is about a jillion lines of C++-ish code, was intended as an interactive GUI application, contains support for multiple different platforms, and the code is, on a scale from a pile of horse shit to not horse shit, approximately a 2.¹⁰ With a medium amount of suffering I got the emulator compiling under mingw in 64-bit mode, and working behind a streamlined interface (`emulator.h`). Once a game is initialized, it is always at an input frame—you can give it an 8-bit input (for the 1st player) to step a single frame, or read the 2048 bytes of memory. You can also save the complete state of the emulator into a vector of bytes, which allows you to restore the state to exactly that same point.¹¹ These save-states are portable across sessions as long as the code was compiled and the game initialized the right way.¹² FCEUX must be single-threaded because it uses global variables galore. I made several enhancements to the emulator interface, which are discussed later.

It's important to know that almost all the CPU time in all the algorithms discussed in this paper is spent emulating NES frames; it takes about 500µs to process a single step. Lots of engineering effort goes into reducing the number of frames the algorithms emulate. The `playfun` program takes a ROM file, the learned objectives and motifs, and runs on the console for arbitrarily long, outputting a movie file consisting of the input sequence it think is best. The current `playfun` algorithm is much too slow to run real-time, but it would be possible to have video output as it played. I disabled most of the video code, however, in an effort to make the emulation loop run as fast as possible.

¹⁰It is, however, an excellent emulator to use, has fancy tools for recording and editing movies, and is popular in the speedrun community. I highly recommend it; just don't read the code.

¹¹This contains the RAM, but also stuff we didn't consider, like registers, the Picture Processing Unit's state, and internal emulator stuff.

¹²The original FCEUX supports portable save-states, but I removed that guarantee in order to get the smallest possible byte vectors. More on that below.

3.2 Naive attempts

The very first thing I tried, as mentioned in Section 2.3, was to just look at all 2^8 different possible inputs at each step, and pick the best one. The inner loop looks pseudolike this:

```
for (;;) {
    vector<uint8> before = GetMemory();
    vector<uint8> state = GetState();
    // Try every bitmask of the 8 inputs.
    for (int i = 0; i < 256; i++) {
        RestoreState(state);
        Step((uint8)i);
        vector<uint8> after = GetMemory();
        double score = Score(before, after);
        // Save the best-scoring i...
    }
    RestoreState(state);
    Step(bestinput);
}
```

`Score` computes a score of two memories using the objective functions, which was the point of all that. There are a few canonical-seeming ways to implement this; the simplest is to count the (weighted) number of objective functions o where `before` $<_o$ `after`. We'll get to more later.

I wish this worked, because that would be truly laughable (and is fast enough to run real-time), but on a scale from doesn't to does it's a 1. At best, Mario twitches in place. The inputs that it plays are insane. There are lots of reasons, but a clear one is that a single input rarely affects your progress in the game on the very next frame. I didn't really expect this approach to work and I already knew that the state space is too big to search exhaustively, which is why I implemented motifs. This drastically reduces the search space and makes each step more significant; the inner loop can now be:

```
for (const Motif &m : motifs) {
    RestoreState(state);
    for (uint8 i : m.inputs()) Step(i);
    vector<uint8> after = GetMemory();
    double score = Score(before, after);
    // Save the best-scoring motif...
}
```

This works much better (anything would), though not much better than you'd expect from just weighted random playback of the motifs themselves (they mostly contain motifs like "hold right" and "hold right and A"). Mario is bad at avoiding danger except by luck, and bad

at jumping hard enough to get over pipes (the button needs to be held consecutively for maybe 40–50 frames to jump that high).

These two things—danger avoidance and microplanning to string together multiple motifs in a useful way—are two sides of the same coin. At least, on a scale from one side of the coin to the other, it is a two. My attempts to address these two problems converged on a single idea that is the crux of the good part of `playfun`. First let's start with avoiding bad futures, since that is somewhat simpler.

3.3 Avoiding bad futures

Scoring a move based on how much better it is than the previous state causes Mario to make sensible greedy moves to increase the objective function—until he is then faced with no good options. This happens very frequently in Super Mario Bros. (and many other games) because death is not usually a single-frame affair. For example, once he's near a Goomba with some velocity, it's too late to slow down or jump out of the way; he'll die in a few frames. Similarly, he can be in the midst of a too-short jump over a pit, where he's destined to die no matter how he squirms. Moreover, in Mario and many other games, even death as recognizable to the player isn't an obvious loss to these objective functions; the game's memory doesn't change much until the interstitial screen and Mario being reset back to the nearest checkpoint. So in order to avoid danger, Mario needs to avoid states that make death a foregone conclusion, not just avoid death.

This is nothing special; move search in Chess and pretty much any game involves evaluating an ending game state and not just the quality of the move itself ("I captured a piece! Must be a great move!"). Evaluating a Chess state is a delicate matter, but Goombas and gravity are very stupid opponents. For avoiding danger, the following works well: Take the state and run a few seconds (300–500 frames) worth of weighted random motifs, several times. This gives us a set of states that could follow our candidate state were we to keep playing. We judge the candidate state not on its immediate value compared to our start state, but based on the random futures that may ensue. In my first version I used the minimum value of all these random futures, so that if Mario was getting into a state where he could die, those states would have low scores. Later we'll find that this isn't the right way to think about it, but it gives us a big improvement in the quality of play—Mario cleanly jumps over Goombas. He also gets very nervous and all like analysis-paralysis when faced

with pits of medium size¹³, which is related to the next section.

3.4 Seeking good futures

The flipside of avoiding danger is seeking adventure. Mario can avoid danger for quite a long time by just waiting at the beginning of the game, until time runs out. He can dilly dally before a pit, contemplating the void. But princesses need rescuin'. The same approach as before works for evaluating a state in terms of its potential: Try some random futures and look at where we end up. We could take the max over those scores if we're being optimistic, or the average or sum if we're trying to judge the general value of the state. In my first version, which did work pretty well, I took the max; so basically I had the min of some random states plus the max of some other states. But why generate a bunch of futures and take the min of some and the max of some others? On a scale of Thank You Mario Your Quest Is Over We Present You A New Quest Push Button B to I'm Sorry, but Our Princess is in Another Similarly-Shaped but Not Exactly that Samesuch Castle, this is an 8.

3.5 Stable futures

Taking the minimum of random futures is always silly (at least if we believe our objective function), because nothing other than our own bad memory can force us to take a series of steps if we know a different better series of steps. Taking the max is possibly foolish if we don't know how to reconstruct a rare good state that caused the maximum score to be high. Both lead us to the idea: Keep around a set of candidate futures that we use for evaluation and search, rather than just randomly generating futures when we want to evaluate a state and then discarding them. This turns out to work really well and be more efficient.

The basic version of the `playfun` algorithm looks like this. Maintain `NFUTURES` futures (this is 40 for the results presented in Section 5), originally just seeded with weighted random motifs. We aren't likely to execute any of these futures verbatim, but they are intended to give us high watermarks of what we're capable of, plus allow us to judge future danger. As we execute search, futures will be partly consumed and extended, and some discarded and replaced.

¹³The companion website contains videos of this, which are funny. <http://tom7.org/mario/>

Each future stores a desired length from 50–800 frames, and whenever it is shorter than that, we extend it (at its end) with random weighted motifs. The inner pseudoloop then looks like this:

```
for (;;) {
    vector<uint8> before = GetMemory();
    vector<uint8> state = GetState();

    set<vector<uint8>> nexts;
    for (Future f : futures) {
        nexts.insert(f.First10Frames());
        f.ChopOffFirst10Frames();
    }

    while (nexts.size() < NFUTURES)
        nexts.push_back(/* random motif */);

    for (vector<uint8> &next : nexts) {
        RestoreState(state);
        for (uint8 i : next) Step(i);
        double score =
            ScoreByFutures(before, futures);
        // Save the best-scoring next...
    }

    ReweightMotifs(best_next, motifs);
    ReplaceBadFutures(futures);
    ExtendFuturesToDesiredLengths(futures);
}
```

At each iteration of the loop we will find the best next 10-frame sequence to commit to. Rather than search over all motifs, we search over the first 10 frames of each future. This has several important consequences. First, since futures are generated by weighted motifs, it makes it more likely that we spend CPU time evaluating motifs that are common; the code from Section 3.2 always explores every motif, even rare ones. Second, it guarantees that if we pick some 10 frames on the basis of a single good future, we don't have to worry about recreating that future; it will still be among our futures for the next round. This is key: It allows us to use the determinism of the game to replay a really good future if we find one, not just use average random futures to assess how good a state will be.

The function `ScoreByFutures` saves the state, then runs each of the `NFUTURES` futures to get a final state and memory. We score each final memory relative to the start memory. The particulars of scoring are not as interesting as the general idea, which is:

- The potential 10-frame `next` sequence that we're

evaluating gets an *optimistic* score. This is based on the futures for which the objectives *go up*. This is always non-negative.

- Each future is also given a score, which is the sum of scores from all the different **next** sequences, regardless of their sign.

The purpose of the first is to identify a good **next** sequence. We take the sequence with the highest optimistic score. The idea is that this is the sequence that gives us the best outcome if we continue to control what we do in the future.

The purpose of the second is to identify which futures are good. A good future tends to bring good outcomes regardless of the immediate next step. Random futures that make us walk left when the good stuff is to the right, or jump when there are spikes nearby above our head, will receive negative scores for many or all of the **next** sequences.

ReweightMotifs changes the weight of motifs that match the best **next** sequence that we just committed to, if we think that we made progress on this step. The idea is to learn which sequences tend to work well for us; this might be different from what the human player did. For example, in run-to-the-right-and-jump kinds of games, human players tend to hesitate more before obstacles and enemies than a computer does.¹⁴ Knowing whether we made progress is kind of difficult, since we can almost always find some move that increases the objectives locally. For this I use the same approach of *value fraction* from Section 2.2 based on a sample of memories that we’ve seen so far. If we appear to be progressing then the motif is weighted up; if we’re regressing then it is weighted down.

ReplaceBadFutures kicks out the the futures with the worst total scores, so that over time the random futures become good futures. Of course, we always have to keep randomly adding to the end of each future, and who knows what crap we’ll find? A late addition to the algorithm replaces some proportion of these with mutated versions of the best scoring future. This helps us from messing up a good future by appending crap to it, since we have multiple copies of it. It also makes search more efficient, since futures that share common prefixes

¹⁴That said, this part was an early idea and is probably not necessary. It’s suspicious because these 10-frame sequences are not necessarily motifs (10 is the same as the normal motif length, and futures are generated by motifs, but they can become desynchronized because of the final incomplete motif, future mutation, and other things). So sometimes the chosen sequence doesn’t affect weights. I think this would be better if we kept a Markov model and updated it no matter what sequences we generated.

can often benefit from caching (Section 4.1). Currently mutating a future involves chopping off its second half (it will get extended to the desired length in the next step), and a $1/7$ chance of *dualizing* the entire sequence. Dualization swaps the left and right buttons, the up and down buttons, B and A, and select and start. The idea of this is to occasionally introduce very different futures to help us get out of stuck situations where we should really be turning around or something like that.

During the development and tuning of the algorithm, I sometimes observed **ReweightMotifs** and **ReplaceBadFutures** conspiring to create a total monoculture of futures, where the weight of a single motif (specifically “press no buttons”) went out of control and all futures just consisted of waiting. Waiting is usually a local improvement to some objectives because of internal frame counters and the cursor used to control music playback. To guard against this, motifs have a maximum (and minimum) possible weight in **ReweightMotifs**, and **ReplaceBadFutures** always ensure that some fraction of the futures are completely random (ignoring motif weights).

Parameter Alert! This is really the worst part in terms of parameters. They are: **NFUTURES**, the number of futures to maintain (40); **NWEIGHTEDFUTURES**, the number of futures that are weighted, as opposed to totally random (35); **DROPFUTURES**, the number of the worst-scoring futures we completely drop (5); **MUTATEFUTURES**, the number of worst-scoring futures that we replace with mutants of the best future (7); **MINFUTURELENGTH** and **MAXFUTURELENGTH**, which put bounds on the sizes of futures (50 and 800); **OBSERVE_EVERY**, the frequency with which we sample memory for computing the value fraction for motif weighting (10); **ALPHA**, the multiplicative factor for up- or down-weighting motifs (0.8); **MOTIF_MIN_FRAC** and **MOTIF_MAX_FRAC**, the bounds on motif weights (0.1 and 0.00001).

Each is a scarlet letter of personal shame and future work is to eliminate them. In my opinion the least excusable are the bounds on future length, since these are related to what portion of time is “interesting” from a game perspective—for example, the max future length must exceed the number of frames that transpire after Mario dies but before he respawns, or the algorithm cannot properly avoid death. In my opinion this requires too much knowledge of the game. I don’t have a

good idea about how to compute this, at least without a human providing an input movie of dying (which would help for lots of other reasons)—but that would complicate the output of the learning process, which violates a primary design goal.

NFUTURES is another bothersome one, but there is more hope here. Basically it trades off computation time for quality of play, and more seems to always be better. We do have runtime metrics that could be used to dynamically change NFUTURES. For example, we could use the notion of improvability from Section 3.6, or the value fraction, the gauge the marginal value of searching an additional future. Or something like that. This might actually help performance because we do the same amount of work (modulo caching) even during periods that the inputs are being ignored, like between worlds in Super Mario Bros., as we do when the going gets rough, and searching for the exact right delicate sequence of inputs would benefit from more options. The appearance of pausing in the output for Bubble Bobble (Section 5.5) suggests that it knows all the futures are bad and needs to search different ones, and corroborates this idea.

3.6 Backtracking

The algorithm above always makes progress at the same rate of 10 frames per iteration. Sometimes it can get stuck in local maxima. Super Mario Bros., my tuning game for this work, seems almost to have been designed to thwart `playfun`. The first level is pretty easy once you have a decent algorithm, and early versions beat it without any trouble. It's mainly about avoiding Goombas and planning far enough ahead to make high jumps over pipes and pits (see the accompanying videos for amusing struggles with these). World 1-2 provides a challenge you may recognize from your youth, immediately followed by something that computers find much harder (Figure 6).

I literally spent about six weekends and thousands of hours of CPU on this problem. First, the overhang is set up so that enemies emerge from it just as you arrive. This means that loose play (imperfect enemy avoidance) tends to get you killed right here. Mario has found a number of different solutions to this problem, from waiting, to kicking turtles from the left to clear out the enemies, to timing his jump perfectly (it is possible!) to stomp the turtle after bouncing his head off the ceiling.

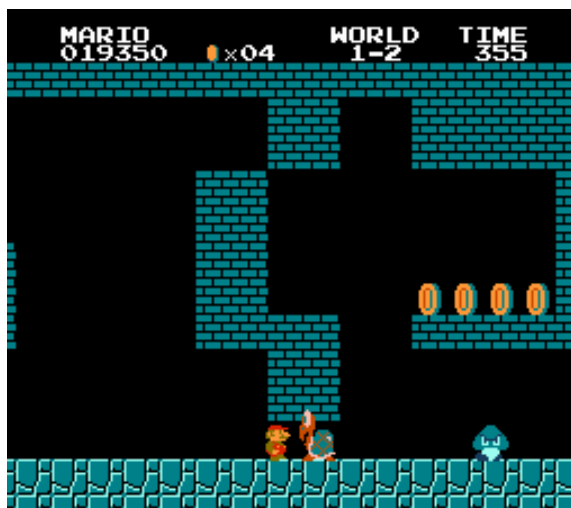


Figure 6: This is where it starts to get hard, even with lexicographic ordering and time travel. This overhang is very tight in terms of the inputs that can get you through it; random play tends to kill you because of the advancing enemies that can't easily be jumped over. Greedy algorithms can't resist the ledge with the coins, but then would need to turn around.

It's fun to see the different approaches and is a good benchmark for whether the search algorithm is actually doing a good job at tactics.

Immediately after this is an important test of longer-term planning. Most of my early solutions would jump up on this ledge with 4 coins, since the score is a component of many lexicographic orderings¹⁵ and coins give you points. Down below is undesirable in the short term because of the enemies. Then Mario would feel stuck and just hump up against the wall jumping in this tiny dead end until time ran out. The game contains some bugs (you have probably seen them) that allow the screen to be scrolled without Mario actually progressing; these little mini scrolls, plus the givens like the frame counter and music cursor, prevent Mario from getting out of the local maximum. This is extremely frustrating. I decided to add some longer-term planning into the search procedure in order to try to help in these kinds of situations, as well as the occasional

¹⁵Maybe even all of them, since it should be globally obeyed; it's therefore a valid extension to any ordering that doesn't already include it. It's not necessarily a tight extension, however, so it may be excluded for that reason, or because during initialization it is not actually zero and so not globally obeyed. I never checked this stuff because I wanted to avoid any temptation to overfit the learning algorithm to the particulars of any game.

deaths I would see.

Every `CHECKPOINT_EVERY` frames we save the state, and then every `TRY_BACKTRACK_EVERY` rounds, we do a backtracking phase if we can. We just need to find a checkpoint at least `MIN_BACKTRACK_DISTANCE` frames in the past. Call that point `start` and the current point `now`. The backtracking routine looks like this:

- Let `improve` be the inputs between `start` and `now`.
- Get `replacements`, a set of input sequences we might use instead. These are usually based on `improve`.
- Add `improve` to the set of replacements.
- Truncate the movie to `start`.
- Now, do the normal playfun loop as though the (truncated) futures are whatever our current futures are, and the set of `next` sequences are the `replacements` array.
- Whatever the best one among those is, keep it. Don't update motifs or futures, however.

The idea is that we take a large sequence from our recent past, and the same futures we're already using, and see if that sequence can be improved, according to our objective functions, and using our same futures. Since the existing sequence is always one of those, if it is the best one, then the backtracking phase is a no-op. If we find something better, we slot it in and continue. So the only risk here is if our objective functions aren't good (we take as given that they are) and the only cost is time.

Generating the candidate set of replacements uses a bunch of different techniques. They are:

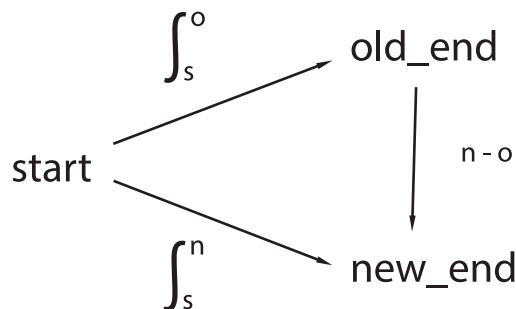
Random. We generate a random sequence of the same length as the `improve` sequence.

Opposites. We dualize (swap left with right, up with down, start with select, and B with A) and/or reverse random spans of the `improve` sequence. The idea is to try to introduce some variety in cases where we're really getting stuck. This was specifically in hopes that Mario would walk off the coin ledge in 1-2 and then find that the course was clear below. I felt okay about this since this seems to be a generic notion (the buttons do have semantics that are common to most NES games where left and right are opposites), but that may just have been rationalization since I was getting desperate. It didn't work; see below.

Ablation. Randomly blanks out buttons from the input. For example, if we don't need to jump and jumping is slowing us down or something, this can remove that and make a small improvement to the sequence.

Chop. Removes random spans from the input. This iteratively removes spans as long as the previous span was an improvement (see below). This can cause the movie to get shorter by removing parts that weren't contributing anything.¹⁶

We use the following formula to score a potential improvement:



The `start` state is as discussed, `old_end` is the state at `now`, and `new_end` is the state after the potential improvement sequence. \int_s^o is the integral of score changes along the `improve` path and \int_s^n along the candidate improvement. The integral is the weighted sum of the objectives increased minus the weighted sum of the objectives that decreased, at each step, all summed up. This works better than just computing the weighted score from e.g. `start` to `old_end` when they are separated by many frames (typical backtrack distances are several hundred frames). The expression $n - o$ ¹⁷ is just the weighted sum of objectives that increased minus the weighted sum of objectives that decreased between the old end state and new end state; we have no option of computing the integral here because we don't have an input sequence that goes from the old end to the new end (and there almost certainly isn't one). We require that three conditions hold:

1. $\int_s^n \geq \int_s^o$,
2. $\int_s^n > 0$

¹⁶However, in games where an objective function includes something like a frame counter or the music cursor, shorter sequences always score lower than otherwise equivalent longer sequences.

¹⁷This is not numerical minus but minus according to the set of objective functions. Just roll with it.

3. $n - o > 0$

The integral has to be at least as good as before, the new integral has to be positive, and the new end state needs to look better than the old end state (the triangle inequality does not necessarily hold here). If they all do, then the score for the improvement is $\int_s^n - \int_s^o + (n - o)$, which is guaranteed to be positive.

Even if we have many possible replacements, we try only the ones with the best scores; due to the particulars of the implementation there is not a simple constant describing this maximum, but it's around 220 with the parameter settings used in this paper.

Parameter Alert! Again! The humiliating appearance of constants! There are many here, having to do with the number of potential replacements we use of each type, the maximum number we keep, how often we backtrack, and how far. I am not totally satisfied with how backtracking works (see below), so I don't want to spend too much energy speculating about how to reduce the parameter space here; I'd rather replace the method wholesale.

The *improvability* of a state is the fraction of these potential improvements that are legal improvements, as above. Based on my analysis, states that are highly improvable ($> 30\%$) tend to be "worse" (more stuck, closer to death, or whatever) than those that are not very improvable ($< 5\%$). This isn't being used for anything currently, but getting an absolute picture of how good a state is (as opposed to simply comparing it to an adjacent state) is one of the difficulties with this approach, so this may be a useful notion for future directions.

Takes of woe, tales of joy. Backtracking was initially added in order to fix the mario coin ledge problem in 1-2. It did not help with this. Older versions of the code would have Mario get past this spot, probably by luck, but in these he would get stuck in an embarrassing way on a small pit later, or jump straight into an enemy. Most of the approaches that actually looked solidly good elsewhere were failing badly here. As I started writing up this paper, on an airplane, I discovered the bug in the weighting of objective functions described in Section 2.2. On my trip I had let `playfun` run with particularly high constants (`NFUTURES = 50`, lots of backtracking candidates, etc.) and it had spent about a thousand CPU hours playing to this point, grabbing the coins, and then dying of timeout, then losing all its lives, starting over, and getting to this point again! After fixing the bug,

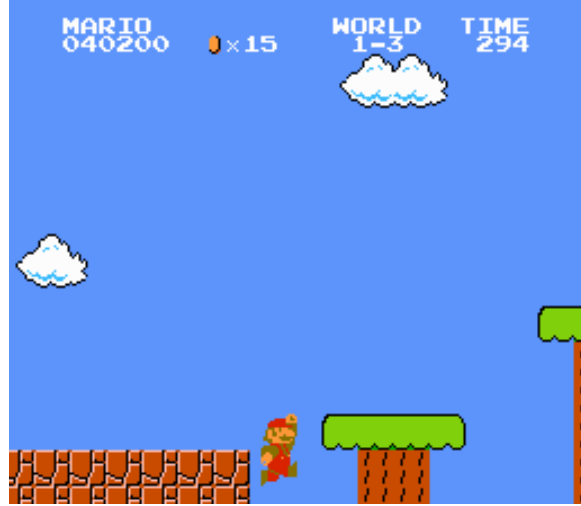


Figure 7: After making me feel so successful by finally cleanly finishing world 1-2, Mario immediately jumps into the first trivial pit in 1-3. I believe what's going on here is that this is actually the best future, because resetting back a single screen isn't a large loss, lives aren't included in the objective function, and probably the rest of the futures were struggling to make progress. This level is very jumpy and probably needs more futures to navigate its obstacles. In any case, Mario, Touché.

I tried again, and it was a huge breakthrough: Mario jumped up to get all the coins, and then almost immediately jumped back down to continue the level! On his first try he beat 1-2 and then immediately jumped in the pit at the beginning of 1-3 just as I was starting to feel suuuuuper smart (Figure 7). On a scale of OMFG to WTFLOL I was like *whaaaaaaat?*

Seeing this huge improvement changed my idea about what part of the code needed work (I now believe that simpler search strategies might work and better lexicographic order generation and weighting is called for). But, this was already in the midst of writing up the paper, so instead I spent the CPU time running the current version on a bunch of games. Those results are described in Section 5 and videos are available at <http://tom7.org/mario/>. In the next section I describe some of the performance improvements I made, and other modifications to the emulator interface.

4 Performance

Performance is really important, both because the quality of the output is CPU-bound and because I am impatient. In its current state, running `playfun` is an overnight affair; it takes about an hour of real time to produce 1000 output frames (16 seconds of gameplay) on a high-end desktop machine. Other than a trick that I didn't end up using, these performance ideas are not at all new, but they are documented here for completeness. I certainly spent lots of time on 'em!

4.1 Caching of emulation

The most expensive part, by far, is running a step of emulation. It takes about 500µs to emulate a single frame, though this can vary depending on what instructions are actually executed (recall that each frame corresponds to many many 2A03 instructions; there are approximately 16,000 clock cycles per frame!). Hardware like the Picture Processing Unit and sound chip are emulated as well, which may actually be less efficient than the actual hardware (for example, the sound hardware can implement filtering with passive physical components like capacitors, but FCEUX contains a finite impulse response filter running on IEEE floating point numbers). We want to avoid executing a step more than once, if at all possible.

Caching is so canonical that some have called it (pejoratively) the only idea in Systems research. And it is what we do here. The emulator adds a call

```
void CachingStep(uint8 input);
```

with exactly the same semantics as `Step`. However, it first checks to see if this exact input has been executed on this exact start state before. If so, it just restores the cached result state instead of executing the step.

I use the excellent CityHash function[4] as the hash code, and use a least-recently-used approach to clean out the hash table when it has more than a fixed *slop* amount more than the target size. States are large because they contain both the savestate of the before and after state. I have plenty of RAM, so I allow each process to store up to 100,000 states with 10,000 states of slop, which takes about 3 gigabytes per process.

Caching adds a little bit of overhead (a single step is about 13% slower, amortized), from saving the state, computing the hash code, copying the state into the table, and cleaning the hash table periodically. A cache hit is a zillion times faster than actually executing the step, however, so as long as the hit rate is more than 13%, it's worth doing. I use caching step in most places

inside `playfun`, except when I know that the state can't have been computed yet. The `playfun` algorithm is particularly suitable to caching: The set of potential next input sequences `nexts` usually share several input prefixes, and we keep around futures for some time, so we can often end up evaluating the same one under the same conditions many times. Moreover, mutant futures usually share their first half, making them half price. A really important property of caching is that it's based on the state of memory but doesn't care about the actual sequence used to get there. This means that in cases where the input is ignored (Mario ignores the jump button most of the time while in the air, for example, and ignores all inputs between levels) we reach equivalent states and can use the cache if the next input is exactly the same. The ideal approach here would be to follow how the bits of the input are actually read by the code, and insert a more generic key into the hash table. For example, if we see that the code never even read the bit of the input corresponding to the up direction, then we can reuse the step for an input that matches all bits but the up bit! This of course requires a lot of mucking around in the internals, which on a scale of within the scope to beyond the scope of this article is a 9.9.

Software engineering lesson: Initial results from caching were disappointing and I thought the overhead was to blame. I made several low-level tweaks to avoid copying, etc., to reduce the overhead from 36% to 13%. Then I discovered that there were 0 cache hits ever, because of a bug (I was inadvertently using pointer equality on keys instead of value equality, so keys were never found). Check basic things about your code's correctness before going on an optimization sortie!

4.2 Space optimizations

Before I had the idea that is the actual subject of this paper, I was playing around with other emulator search ideas that involved storing very very large numbers of states. This necessitated minimal representations of savestates. FCEUX uses zlib internally to compress states, which works well; they shrink from something like 13,776 bytes¹⁸ to an average of 2266. I made some modifications to the save/load routines to make this as small as I could manage. I removed the backing buffer,

¹⁸I don't completely understand the NES architecture and emulation strategy, but I believe some games include expansion chips that require more space for save states. All this work is based on the 2048 bytes of main NES RAM, as you already know. Perhaps clever video game authors from the past can protect against this paper's approach by storing important game facts inside expansion chips in the cartridge.

which is only used for drawing graphics, movie info, header bytes that are constant or already known (savestate size), which shrunk the compressed size to an average of 2047.42 bytes. That meant I could fit about 32 million savestates in RAM at once, which would be kind of amazing if I could tell my 8 year-old self that.

Compression algorithms work well when there is lots of redundancy in their input. Comparing memories between two frames in some game, you'll often see only a few differences. Some of the memory contains stuff that's essentially read-only, even. To improve the compressibility of savestates, I introduce the idea of a *basis* state. This is any non-empty sequence of bytes which is an additional argument to the `LoadState` and `SaveState` functions. It's subtracted from the savestate prior to compression (it doesn't have to be the same size; it's treated as an infinite repetition of those bytes). If the basis is the same as the savestate, then the result is all zeroes, which compresses nicely (of course, now you need to have the basis in order to load the state, which is the same as the savestate, so that didn't save you much!). But memories during gameplay often don't differ much, so I save a state from any arbitrary frame in the game and then use that as the basis for everything else. This factors out any common parts and makes the states much more compressible: The average size drops to 1870.41 bytes.

Using a better compression algorithm like the Burrows-Wheeler transform[3]¹⁹ would probably help too; zlib implements LZW which is only mediocre. However, on a scale of throwing my computer out the window to wanting to reimplement all software in my own private SML library, I just can't stand getting 3rd-party libraries to compile and link into my applications, so I didn't even try. In any case, I abandoned this direction, which was not working well even with lean savestates.

For `playfun`, RAM is not a bottleneck at all. I disable compression completely on savestates, including the builtin zlib stuff (which is actually rather slow) and just use raw savestates. The removal of unnecessary headers and stuff is still there, and saves a few bytes and CPU cycles, but there was no reason to actually do it for this particular work. *But sometimes I do unnecessary things.*

4.3 MARIONET

The `playfun` algorithm involves running 40 or so different 10-sequence `next` steps and then scoring them

¹⁹This is really one of the all-time great algorithms. If you don't know it and you like this kind of thing, you should check it out. It's so surprising and elegant.

against `NFUTURES` different futures. Each `next` and each future is completely independent and dominated by the cost of evaluating emulator steps. The ideal thing would be to use threads and shared memory to run these steps in parallel. As I mentioned earlier, `FCEUX` is hopelessly single-threaded.

Eventually I realized I needed to search more states than I could in a single thread, and so I created `MARIONET`. It's a play on words, a double entendre of "Mario network" and "Marionette", which is obvious. This is a mode whereby `playfun` can be started in "helper" mode (listens on a port for work to do) or "master" mode (connects to a list of ports to run work on), in order to run many emulators in different processes.

The processes communicate over TCP/IP. I only run it on my local computer, but it would be reasonable to run it on a cluster or possibly even the Internet. I used SDL's `SDL_Net`[6] as a portable network interface in the hopes of keeping it platform-agnostic (`FCEUX` is basically portable so it would be a shame to make this Windows-specific, though I am not eager to try to get this thing to compile on other platforms, I gotta be honest). SDL is a nightmare to get working with mingw in a 64-bit compile, as usual, and `SDL_Net` contained some bugs²⁰ that I had to spend some time working around. Anyway, I'm just complaining. For serializing requests and responses to bytes, I used Google's Protocol Buffer library[5], which is designed for this kind of thing.

Helpers all start up with the same ROM and motif and objectives loaded, so that they can simulate whatever the master would compute.²¹ They just wait on a port for the master to send a request, which can be either "try this set of `nexts` with these futures and send me the results" or "try to find n improvements of these inputs `improveme` using so-and-so approach." In either case we send the base state as a serialized savestate.

The master does the outer loop of the algorithm and all the writing to disk and so on. When it needs to do an expensive step like the inner loop, it prepares a set of jobs and sends them to workers using a little fork-join library (`netutil.*`). The library keeps track of the outstanding jobs and which helpers are working,

²⁰In particular, the `SDLNet_TCP_Recv` call is supposed to block until it has received the entire requested length, but it occasionally returns early.

²¹Only the master reweights motifs, since it needs a single sample of memories that we've observed. That means that in cases where a helper generates random motifs, it does so with respect to the original human weighting. There are some other small things I simply avoided doing because they would require too much coordination between processes or make the RPCs too large; `MARIONET` was part of the system for much of its development.

and feeds them work when they are idle, until all the jobs are complete. It’s even got a color ANSI progress bar and stuff like that.

Utilization is very good (Figure 8), and we get almost a linear speedup with the number of processes, up to the number of hardware threads the system supports (twelve). Actually, in addition to the computer remaining usable for other stuff, $n - 1$ helpers may work a bit better than n , because each of the helpers is then able to avoid preemption. Since the driver loop is synchronous, having one laggard process slows the whole darn thing down while everything waits for it to finish.

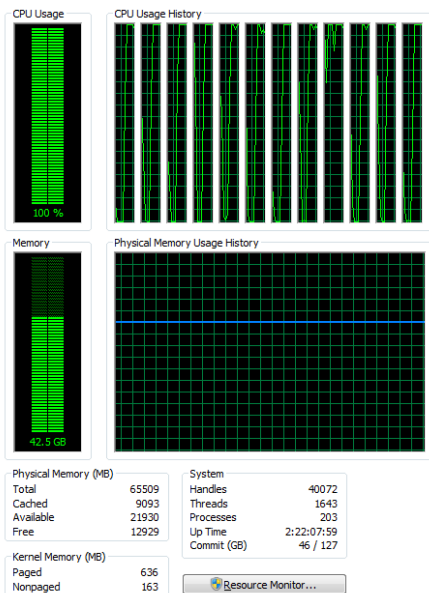


Figure 8: Utilization with 12 helpers and one master on a 6-core (12 hyperthreads) processor. Looks good. Keeps the bedroom warm.

MARIONET is a huge improvement and was what enabled using 40+ futures in a reasonable amount of time, which is key to high-quality output. It’s obviously the way you want to run the software, if you can get it to compile. There is one downside, though, which is that it reduces our ability to get lucky cache hits across `nexts` that are similar (or have the same effect), and when replaying the winner as we commit to it. It’s worth it, but smartly deciding which helper gets which tasks (because they share prefixes, for example) would save time. Running it all in one process with a shared memory cache would be the best, as long as the lock contention could be kept under control.

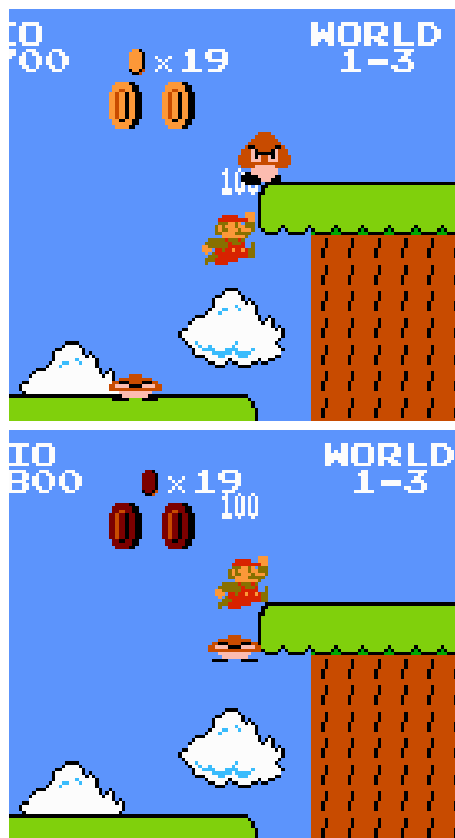


Figure 9: Mario bounces off one Goomba up into the feet of another. Not only doesn’t he have enough velocity to reach the platform, but he’s about to hit that Goomba from below. Believe it or not, this ends well: `playfun` is happy to exploit bugs in the game; in this case, that whenever Mario is moving downward (his jump crests just before the Goomba hits him) this counts as “stomping” on an enemy, even if that enemy is above him. The additional bounce from this Goomba also allows him to make it up to the platform, which he wouldn’t have otherwise!

5 Results

In this section I describe the results of running `learnfun` and `playfun` on several NES games. These games were all played with the same settings developed and tuned for Super Mario Bros.; that is, this was the result of just running the program the first time on a recording of me playing the game briefly. Every game I tried is documented here; lack of CPU time before conference is the main reason your favorite is not included. The web-

site <http://tom7.org/mario/> contains video versions of some of these, which may be more fun. Let’s begin with the *ur* game, Super Mario Bros..

5.1 Super Mario Bros.

This game is an obvious benchmark and the one I used during the development of `learnfun` and `playfun`. Some parts of the algorithm (see e.g. Section 3.6) were developed specifically to get around problems in the ability to play this game, though I believe simpler methods would also work on this game, now that some important bugs are fixed.

Automated Mario is fun to watch, and definitely my favorite. The familiarity of the game, the combination of human-like maneuvers and completely bizarre stuff, daredevil play, and bug exploitation (Figure 9) are all a delight. It’s even funny when it fails, watching Mario struggling with obstacles like tiny pits, or making a heroic move followed by a trivial mistake. I recommend watching the videos.

This algorithm works well on Super Mario Bros., and I think that with some improvements it could play quite far into the game. It probably can’t beat the game; Worlds 7-4 and 8-4 require some weird puzzling that we all needed *Nintendo Power Magazine’s* help to solve as kids, and are not just a matter of running to the right and avoiding obstacles. In the current incarnation, Mario breezes through World 1-1 every time, consistently beats 1-2 (I’ve only tried about three times with the newest version of the code, but each time he’s succeeded) and has limited success on 1-3 but eventually suicides too many times. I’ll keep trying.

The Mortal Kombat-style *Finish Him!* to any machine learning algorithm is overfitting, however. Does the technique work on other games, without endless tweaking as I did with Super Mario Bros.? On a scale of no to yes, this is both a 1 and a 10; some games work even better than Mario and some are a disaster.

5.2 Hudson’s Adventure Island

This is a bad, difficult, but likable game featuring a skateboarding cherubic island guy called Master Higgins, whose girlfriend has been kidnapped by King Quiller. He basically runs to the right and can throw or ride stuff that he finds in eggs. The controls are pretty soft and danger is everywhere. The objective functions that work are probably pretty similar to Super Mario Bros., but there are fewer obstacles to navigate—the difficulty for humans mostly comes from the speed and reaction time. Master Higgins doesn’t care about

bumping into rocks and dropping his health almost to nothing (but then is careful about health), and once he gets a weapon his aim anticipates off-screen enemies and he looks pretty savvy (Figure 10). His weakness regards holes in the ground, which he sometimes jumps into, probably for the same reason that Mario sometimes does. His “pot bonus” was 16720. Master Higgins beats several levels and makes it into the cave before losing his last life by jumping straight into a vibrating skull with swirling fireballs around it, which on a scale of Darwin Award to dignified demise is approximately a 6, in my opinion.



Figure 10: Master Higgins putting safety first.

5.3 Pac-Man

One of the smallest NES games at a mere 24kb, Pac-Man is a classic that I won’t belabor. It has a fairly straightforward objective—your score—and there is no timer or anything to worry about but the ghosts. A good planner could probably solve Pac-Man with just score as an objective (keep in mind that with RAM inspection and because of ghost movement, this is not just a simple matter of using A* or something to plan a path through the Euclidean grid—states aren’t repeated hardly ever). Automating this game works pretty well; Pac-Man doesn’t have much trouble avoiding ghosts except when they trap him. Play is pretty strange: He doesn’t necessarily grab nearby dots (hard to explain) and often switches direction in the middle of a corridor, unlike human play which is usually efficient sweeps of the dots when ghosts aren’t near. However, automating

has a huge advantage over human players with respect to ghosts, and Pac-Man is alarmingly fearless. He chases ghosts down corridors, turns straight into them, knowing that they will change direction in time to not run into him (this makes the time travel advantage quite stark). Figure 11 shows gratuitous daredevilism as he ducks in and out of a tiny sliver of space between two ghosts, then does it again, and survives.

Eventually, Pac-Man gets far enough away from the remaining dots that none of his futures bring him near, and without any other objective to seek out, runs into ghosts. He dies on the first level with 13 dots left.

5.4 Karate Kid, The

Karate Kid, The, is the typical trainwreck that follows when a beloved movie is turned into a video game. The game has awful controls and integer-only physics with massive throw-back upon collision, strange mini-games with no explanation, annoying debris flying through the sky, and luck-based fighting. It was probably only ever finished by kids with extreme discipline and self-loathing, or kids with only one video game available to them. It begins with a karate tournament which is even less fun than the main game, which is sort of a platformer with very few platforms.

In this game I played 1,644 frames, just the first two of four opponents in the karate tournament. Automated by `playfun` Daniel-San is able to punch and kick the low-level karate noobs, preferring to spend all his super-strong Crane Kick power moves right away. His style doesn't make much sense, sometimes facing away from the opponent, or throwing punches and kicks when the opponent isn't near. He doesn't worry much about taking damage. He gets to the final round and makes a valiant effort, at one point taking himself and the karate boss to 0 health simultaneously. But in this game, tie goes to the computer-controlled player without feelings, so it's back to the title screen for Daniel-San. The result is not impressive at all; the main goal here is to reduce the opponent's health, but our objective function can only track bytes that go *up*. Still, the automated version gets farther than my input did.

5.5 Bubble Bobble

Let's take a journey to the cave of monsters! This lovely game features two stout dinosaurs called Bub and Bob, who jump around in a series of single-screen caves. You can shoot bubbles to encase monsters and then pop the bubble to turn them into fruit or other treasure; clearing all the monsters takes you to the next stage. Bubbles

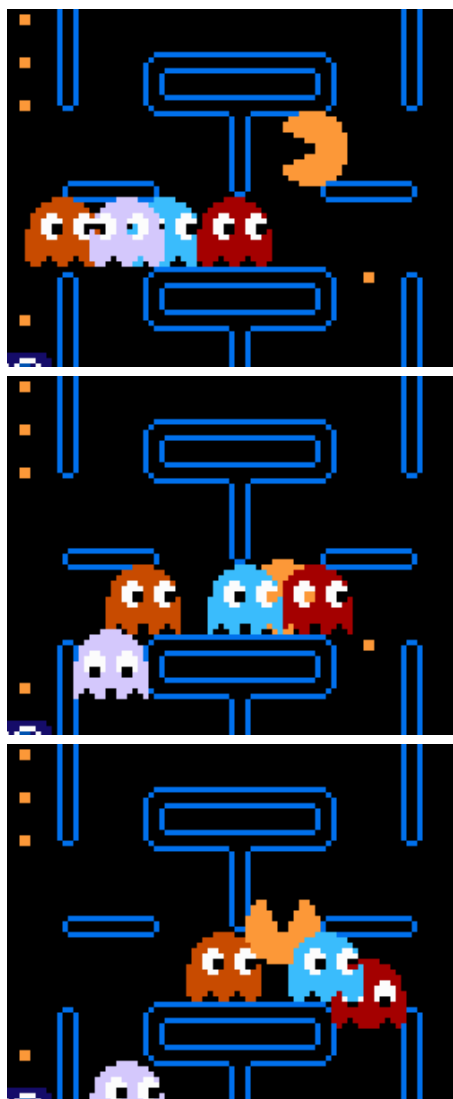


Figure 11: Pac-Man showing utter disregard for the ghosts's personal space. This occurs around frame 6700 of the output movie. Pac-Man slips into the space between Blinky and Inky to touch the wall, comes out unharmed, then momentarily teases Clyde before escaping to vibrate some more in empty corridors.

are also necessary for navigating around, since you can bounce on your own bubbles when holding the jump button.

I was surprised that automating this game works at all, since there is no obvious spatial gradient like in Super Mario Bros., and few local greedy rewards like



Figure 12: Daniel-San blowing his last Crane Kick on a karate noob like there’s no tomorrow, which there won’t be if he uses up all his power moves like that. Note that the **Chop** approach to backtracking was used to generate this movie frame, appropriately.

in Pac-Man. Bub initially wastes his lives (a common theme in games where the respawn interval is low—it just looks like a fast way of warping to the corner). When he’s near monsters he shows good tactics in shooting and popping them simultaneously, even doing so while facing away from them, which I didn’t know was possible. At the beginning of the movie he prefers to hide in the bottom-left corner, but soon starts jumping all around the level, bouncing on his own bubbles and so on, almost like a real player would. He beats more levels than my input movie did! Since it takes two start button presses to enter the game, the second START is part of the input motifs. Amusingly, Bub uses this to his advantage to change the synchronization of his futures, and when the situation gets really tight, he’ll pause the game until a good future comes around (Figure 13). Paused states probably look like modest progress since memory locations like the music cursor are still advancing.

After a harrowing escape from the ghost thing on level 4, Bub makes it to level 5 (my input movie quits at the beginning of level 4!), at which point I stopped the search because it was getting pretty pausey and I wanted to see some other games before the SIGBOVIK deadline.



Figure 13: Bub navigates this game surprisingly well. Once he’s on his last life, he becomes careful and pauses the game when things are looking grim—here pausing for about a thousand frames, burning through the futures until one randomly comes along that looks good. He then unpauses and executes that good future, killing three of these monsters in short order.

5.6 Color a Dinosaur

This is a strange NES “game” where you do the eponymous action, probably intended for kids. It’s not very fun to watch the weird slow floodfill algorithm color in parts of a dinosaur, and there’s no objective to be found. Predictably, the approach of this paper doesn’t simulate human play very well. There doesn’t even seem to be an objective for humans to suss out, except for the open-world sandbox of two-color dinosaur coloring. The automated pencil manages to “color” two different dinosaurs (Figure 14), though the play looks pretty spastic and random.

5.7 Tetris

Tetris is a block dropping game, also known to the ancients. The Nintendo implementation is infamous for being inferior to the unlicensed Tengen version, which Nintendo tried to sue out of existence. Here I try to automate the Nintendo version as a tribute to litigation. Unsurprisingly, **playfun** is awful at this game. Tetris has several screens of menus, where the player can select between different modes and theme musics and stuff like that. As an amusing prelude to this disaster in tetromino stacking, **playfun** keeps entering the menu and

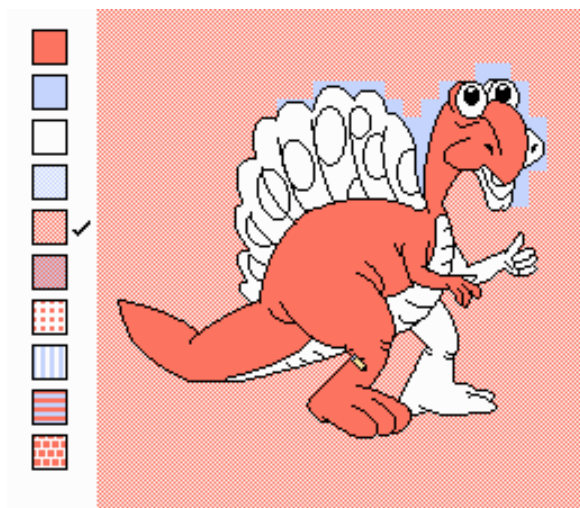


Figure 14: The second dinosaur colored by the `playfun` algorithm. There’s no way to go wrong in this game; any possible coloring is *oh so right*.

exiting back to the title screen rapidly, taking several seconds to even start the game. (Like in *Bubble Bobble*, this means that the start button is among the motifs.) Although the piece dropping looks more or less natural (but it’s hard to not be, as the game drops the pieces for you), the placement is idiotic—worse than random. This may be because placing a piece gives you a small amount of points, which probably looks like progress (Figure 15), so there is incentive to stack the pieces as soon as possible rather than pack them in. As the screen fills, there’s a tiny amount of tetris-like gameplay, probably as the end of the game becomes visible in some of the futures. The end result is awful and `playfun` makes zero lines and certainly no Tetrises (Figure 16). The only cleverness is pausing the game right before the next piece causes the game to be over, and leaving it paused. Truly, the only winning move is not to play.

6 Future Work

It is famous last words, but I actually intend to keep working on this project. Because of SIGBOVIK crunch pressure (and discovering some bugs as I was writing the paper) the approach got much better very recently and I’m mostly limited by CPU hours until conference. It’s today in a state where whenever I run it on a new game I see something delightful. Even just running it on more of the NES classics is worthwhile. However, I have

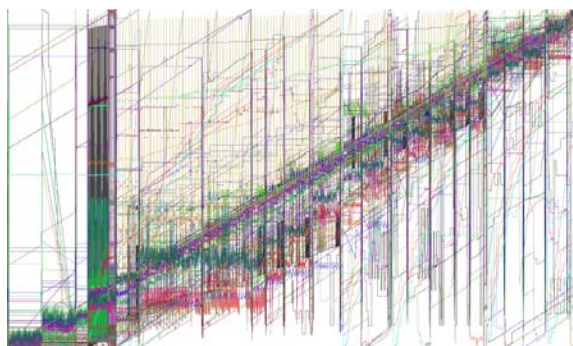


Figure 15: The joyful noise of objective functions learned for Tetris. The first fifth of the movie is navigating menus and looks very different from the remainder. There appear to be some frame counters identified, which make perfect smooth progress throughout the entire movie. I believe discontinuities correspond to placing of pieces; some objectives monotonically increase at these points (and are noisy in-between), suggesting that they incorporate the score.

lots of ideas about how to extend the technique, either to make it more beautiful or to make it work better:

Parameter reduction. I’ve already pointed out the places where there are mysterious constants in the code. Although the fact that the same constants work on many different games without tuning is some solace, it would really be nicer to remove these. One of the hardest to remove is the length of the futures explored. And I like a challenge!

Unsupervised learning. Speaking of challenge, it might be possible for `playfun` to teach itself, by starting with no objective functions and playing randomly to see what bytes it *can* make go up, then fitting lexicographic orderings to the memories and repeating. The beginnings of games (which include RAM initialization, title screens, and menus) continue to vex this kind of work, unfortunately.

Generalized orderings. Right now, lexicographic orderings are limited to vectors of unsigned bytes that get larger. Many games treat bytes as signed (I believe this is true of Mario’s velocity, for example). For other bytes, like one representing your enemy’s health (c.f. *Karate Kid*), winning means making the byte *go down*, not up. It is possible to generalize the lexicographic orderings we generate to a vector of $(L_i, <_i)$ pairs, where

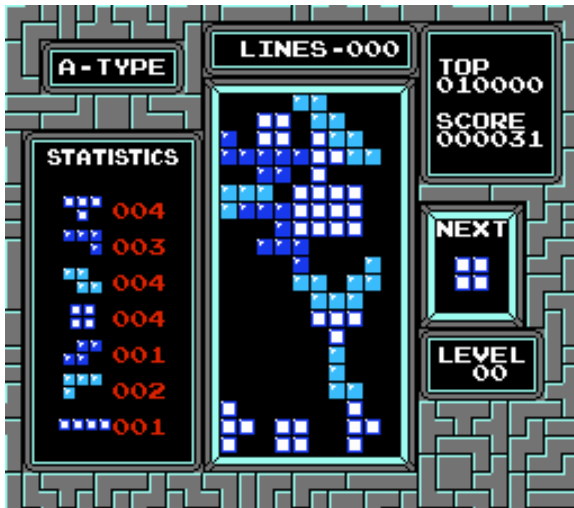


Figure 16: Would you hire this construction company? Death is imminent, so `playfun` pauses the game shortly after this and then doesn't unpause it.

$<_i$ tells us how to compare the bytes at that location. Things to try would be two's-complement signed comparison, or unsigned greater-than. I think this is a great avenue; the dangers are overfitting (now most short sequences can be explained in one way or another) and being too fancy.

Input models. I'm unsatisfied with the motif approach. As mentioned earlier, the obvious thing to try instead is a Markov model, which would probably be simpler and would allow re-weighting from inputs regardless of what we concocted while playing (the current version can only reweight the human motifs, not learn new sequences discovered while running). I would also like some solution to the start button problem—if it is among the motifs, it often shows up in gameplay in annoying ways. I don't feel good about simply blacklisting it, however. In *Bubble Bobble*, start appears to be used to burn away futures when all of them are bad. Maybe a simple improvement would be to allow the inner loop of `playfun` to skip the turn (empty `next` sequence) in order to simulate this.

Better backtracking. Backtracking is a powerful idea, and there's lots more to do here. The fixed-size backtracking window is disappointing since it's a parameter, and doesn't even seem right for most games. It would make sense to do something like lengthen the window until the *improvability* starts dropping; basically,

find the amount of recent past that's comparable to random play, and try improving that. Moreover, it should be possible to re-backtrack, since some parts of the game are harder than others. Ideally, when Mario has so few options that he contemplates suicide, he should be backtracking over and over and farther and farther until he finds a future that's good. This is the power of time travel. Use it, man.

Efficiency in search Quality is directly related to the number of states searched. Some sequences are easier to search than others, because they share a prefix with something we've already done and can be cached. It would be worth looking into algorithms that explicitly take into account the cost of branching, so that we explore some tree (of futures, for example) rather than disjoint linear futures. The effective number of futures explored could be much higher for the same CPU.

Multiple players, multiple games. Other than the particulars of the way it's built (`vector<uint8>` everywhere), there's no reason why the technique is limited to a single player's input. In a game like *Bubble Bobble* or *Contra* the two players can collaborate in interesting ways, and planning both simultaneously would probably lead to occasional awesome results. For example, in *Contra*, it might be that one player is shooting enemies for the other player, the bullets arriving from across the screen just in time to save him as he blithely jumps into danger. Another clever feat from the Tool Assisted Speedrun community is a sequence of inputs that beats multiple different *games* simultaneously. For example, human geniuses used tools to beat *Mega Man* (*Mega Men*?) 3, 4, 5, and 6 all at the same time using the exact same input sequence sent to all four games.[2] I think the algorithms in this paper apply directly—it's just a matter of multiplexing the inputs to each of the games, and then taking the appropriate operation (min, max, sum) of their objective functions to produce a final objective function. The main obstacle is the architecture of the underlying emulator, which can only load one game into its global variables at once.

7 Conclusion

In this paper I showed how lexicographic orderings and time travel can be used to automate classic video games. The approach uses an amusingly simple and mathematically elegant model to learn what constitutes "winning" according to a human player's inputs. It then uses hundreds of CPU hours to search different input sequences

that seem to “win”, inspecting only the RAM of the simulated game, ignoring any of the human outputs like video and sound. The technique works great on some games, getting farther into the game than the human’s inputs did, and produces novel gameplay (for example, bug exploitation and super-human daredevil timing). On other games, it is garbage, but on a scale of recycling symbol 1 to recycling symbol 7, it is at least hilarious garbage.

References

- [1] adelikat et al. FCEUX, the all in one NES/Famicom emulator. <http://fceux.com/>.
- [2] Baxter and AngerFist. NES Mega Man 3, 4, 5 & 6. <http://tasvideos.org/871M.html>.
- [3] Michael Burrows and David Wheeler. A block sorting lossless data compression algorithm. Technical Report Technical Report 124, Digital Equipment Corporation, 1994.
- [4] gpike and jyrki. The CityHash family of hash functions, 2011. <https://code.google.com/p/cityhash/>.
- [5] kenton, jasonh, temporal, liujisi, wenboz, and xi-aofeng. Protocol buffers, 2013. <https://code.google.com/p/protobuf/>.
- [6] Sam Latinga, Roy Wood, and Masahiro Minami. SDL_net 1.2, 2012. http://www.libsdl.org/projects/SDL_net/.
- [7] Vargomax V. Vargomax. Generalized Super Mario Bros. is NP-complete. *SIGBOVIK*, pages 87–88, April 2007.