how to `git bisect` when your test suite is having a bad day

ben blum

bblum@alumni.cmu.edu

abstract

I like probability puzzles but don't know enough stats to actually solve them properly so I threw some thinking sand at it and learned some interesting stuff anyway.

keywords pdflatex, cdflatex

1. problem statement

Let's say you're trying to bisect a given range of n commits.[1] Call them $c_0 \dots c_{n-1}$, where c_0 is known safe and c_n is known to have the bug. You'd probably start by testing $c_{n/2}$, right? And you'd expect to pinpoint the buggy comit in $\log(n)$ steps. That's math.

Ok, but what if the bug reprodues nondeterministically with some probability $p < 1$. You can't even pinpoint the bug at some c_b for sure anymore; you can at best know that it ~prooobably~ won't reproduce in any $c_{i<b}$, with some confidence z. Now evaluate your strategy by the expected number of steps to achieve, let's say for tradition's sake, $z > 0.99999$. Is it still optimal to bisect at the midpoint?[2] What would be a better strategy, as a function of p?

2. intermission

Put the paper on pause now and think about it. No, really, give it a go! I mean, if you don't think math puzzles like this are cool, just stop reading, no worries. I'm sure there's a paper about, like, hygienic image macros or empire machines or something for you just a few page-turns away.

[1] To use binary search to find a commit that introduced a bug.

[2] Spoiler: No, or I wouldn't have bothered with this paper.

sigbovik '20 pittsburgh, PA, USA

ACH ...$13.37

If you're feeling adventurous, implement a strategy and throw it in this simulator I made to see how it fares: `https://github.com/bblum/sigbovik/blob/master/bisect`. You just gotta implement `trait BisectStrategy`, and it even does all the hard work of applying Bayes's rule for you and letting you see the PDF and everything. Check it out.

3. pdfs, but not the portable document format kind, and our friend rev. bayes

Ok, so let's model the problem as a sequence of steps on a probability distribution function (henceforth, PDF; and also, CDF for the cumulative kind). Initially, $\mathsf{pdf}(i) = 1/n$ for all $0 \le i < n$. When the bug reproduces at some commit b, you're certain no $c_{i>b}$ introduced the bug, so each $\mathsf{pdf}(i > b) \leftarrow 0$ and each $\mathsf{pdf}(i \le b)$ renormalizes by a factor of n/b, to preserve the $\sum_i \mathsf{pdf}(i) = 1$ invariant.[3]

In the deterministic case, $p = 1$, the symmetric thing happens when the test passes at some c_j: $\mathsf{pdf}(i \le j) \leftarrow 0$. But when $p < 1$, we must generalized it (Vargomax 2007). Here's Bayes's rule:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

In this case, B is that the test passes at c_j, and A is that bug exists at or before c_j after all. $P(B|A)$ is the false negative rate, i.e. $1-p$. $P(A)$ is the prior on c_j containing the bug, i.e., $\mathsf{cdf}(j)$. And $P(B)$ is the false negative rate weighted by the bug existing, i.e., $(1-p)\mathsf{cdf}(j) + 1(1-\mathsf{cdf}(j))$. To update our priors on commits up to j, we renormalize by dividing out the old $\mathsf{cdf}(j)$ and multiplying by the new $P(A|B)$, i.e.,

$$\forall i \le j, \mathsf{pdf}'(i) \leftarrow \mathsf{pdf}(i) \frac{1}{\mathsf{cdf}(j)} \frac{(1-p)\mathsf{cdf}(j)}{(1-p)\mathsf{cdf}(j) + (1-\mathsf{cdf}(j))}$$

[3] Implemented as `fn adjust_pdf_bug_repros()` in the simulator.

Which simplifies to:

$$\forall i \le j, \mathrm{pdf}'(i) \leftarrow \mathrm{pdf}(i)\frac{1-p}{1-p\,\mathrm{cdf}(j)}$$

Call this renormalization factor $\mathcal{R}$. As a sanity check, $p\,\mathrm{cdf}(j)$ is less than p, so $\mathcal{R}_{i\le j} < 1$.

Likewise, for commits above j, we have $P(B|A) = 1$, $P(A) = 1-\mathrm{cdf}(j)$, and $P(B)$ the same as before. Renormalizing from $1-\mathrm{cdf}(j)$ this time (and skipping the unsimplified version), we get:

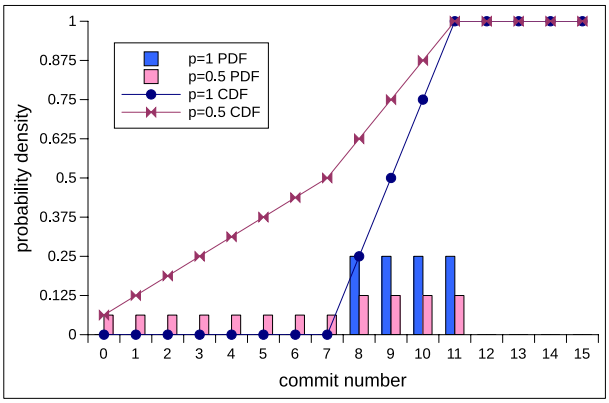
$$\forall i > j, \mathrm{pdf}'(i) \leftarrow \mathrm{pdf}(i)\frac{1}{1-p\,\mathrm{cdf}(j)}$$

As a sanity check, $p\,\mathrm{cdf}(j)$ is positive, so $\mathcal{R}_{i>j} > 1$. If you like pen-and-paper algebra, you can also see that $\mathrm{cdf}(j)\mathcal{R}_{i\le j} + (1-\mathrm{cdf}(j))\mathcal{R}_{i>j} = 1$.[4]

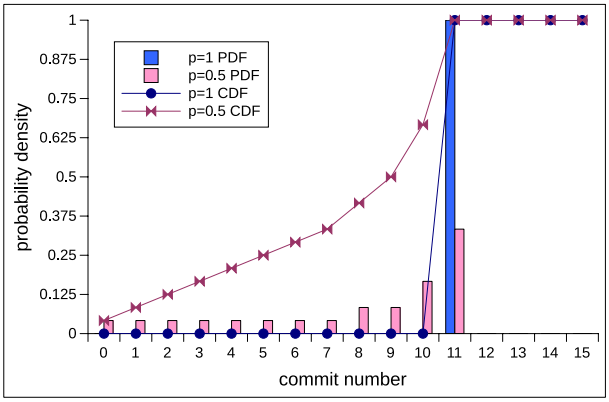
Let's do a nice concrete example. Say $n = 16$, the test passes at $j = 7$, and then fails at $j = 11$. In the deterministic case, all the probability mass will be concentrated uniformly in the range $[8, 11]$. However, if the bug repros only half the time, $\mathcal{R}_{i\le 7} = 2/3$ and $\mathcal{R}_{i>7} = 4/3$, and we get probability mass scattered all the way down to c_0, as shown in figure 1(a). Yuck, someone clean that up!

Now let's say the test passes at $j = 9$, then at $j = 10$. figure 1(b) shows the updated PDFs/CDFs: for $p = 1$, this pinpoints the bug at $j = 11$, and the search is over. But for $p = 0.5$, there's still 2/3 odds we'd be wrong! In fact, from here it takes 18 further probes at $j = 10$ until we are at least five 9's confident that c_{11} is the culprit.[5] Bayes's rule gonna get ya.

A noteworthy invariant here is that the PDF is always monotonically nondecreasing in its nonzero range: each passing test always shifts probability mass to the right of the bisect point, but past the earliest known bug repro, nothing can ever revive it back above 0.



(a) after two tests



(b) after "classical" search terminates

figure 1. example {,non}deterministic {P,C}DFs

4. prior work

I was kinda surprised to find no existing mathy solution to this problem lying around on the online. Wikipedia has a brief little subsection on "noisy binary search", which links a few papers older than I am. In one (Rivest et al. 1980), they bound the number of erroneous answers by a fixed factor of the number of queries, so it's more like "twenty questions with lies" than bisect. In another (Pelc 1989), they do fix the error rate p, but they allow for symmetric false negatives and false positives, both with the same p. This too changes the nature of the problem; notably, if $p = 0.5$, you can't make any progress whatsoever.

Dropbox has a CI service called Athena (Shah 2019) which automatically searches for flaky tests. In this case the goal is to keep the build green, but if you consider the flaky test itself to be the bug, it's the same problem.[6] Athena "deflakes" the test at each commit by running it 10 times, treating the combined result as "basically as good as $p = 1$", and then runs an otherwise classical binary search. In this setting, p is not known in advance, so using Bayes's rule

[4] Implemented as `fn adjust_pdf_no_repro()` in the simulator.

[5] See `fn test_figure_1()`.

[6] Incidentally, the symmetric case – where a bug repros with $p = 1$, but the test also flakes with some $q < 1$ – is also the same problem.

is off the table. But I will show that even without access to the PDF, a better strategy exists.

5. strategies

Ok, so how do we make progress, i.e., concentrate probability mass til there's z of it in one place, as quickly as possible? Let's deconstruct the motives of classical binary search. Let c_b denote the earliest known buggy commit, and c_a be the latest known safe commit. In Determinism World, bisecting at $c_{(a+b)/2}$ minimizes the worst case maximum range remaining, as the two possible outcome ranges are the same length. But in Nondeterminism World, a doesn't ever budge from 0, so bisecting at $c_{(0+b)/2}$ will not even terminate. Sure, hitting the PDF with $\mathcal{R}_{b/2}$ will always move *some* mass rightward, but once five 9's of it is already over there, it can't concentrate it onto one point. So let's not think about the range.

5.1 bisect probability mass

Another way to frame it is that the $c_{(a+b)/2}$ bisect point cuts the probability mass, rather than the range, in half, i.e., $\mathsf{max}_j(j;\mathsf{cdf}(j)\le 0.5)$. This approach fits the "binary search" spirit, and will also terminate in Nondeterminism World: if b is the solution, it converges to repeatedly probing $b-1$, so it can pass any fixed z threshold. But is 0.5 still the best bisect point even when $p<1$? I wondered if this could be expressed as the amount of probability mass moved from one side to the other, i.e., $\sum_i \mathsf{abs}(\mathsf{pdf}(i)-\mathsf{pdf}'(i))$. In the case where the bug repros this is:

$$p\,\mathsf{cdf}(j)\times(1-\mathsf{cdf}(j))$$

and in the case where the test passes:

$$(1-p\,\mathsf{cdf}(j))\times\mathsf{cdf}(j)\times(1-\mathcal{R}_{i\le j})$$

which, surprisingly, simplifies to exactly the same thing as the bug repros case. The maximum occurs where $\partial/\partial\,\mathsf{cdf}(j)$ is 0, which turns out to be at $\mathsf{cdf}(j)=0.5$ after all, and independent of p.[7] But it's not clear that the amount of mass moved necessarily corresponds to reaching z the fastest. I show my work in figure 2, because that's what they taught me to do in high school algebra class.

[7] I also worked out the *minimum* moved mass between the pass and fail case, which is almost always the pass case. It comes out to $\frac{\mathsf{cdf}(j)(1-\mathsf{cdf}(j))}{1/p-\mathsf{cdf}(j)}$, which experiences its maximum at $\frac{1-\sqrt{1-p}}{p}$ (thanks wolframalpha). But this is worst-case thinking, which doesn't seem appropriate. Let's have some optimism!

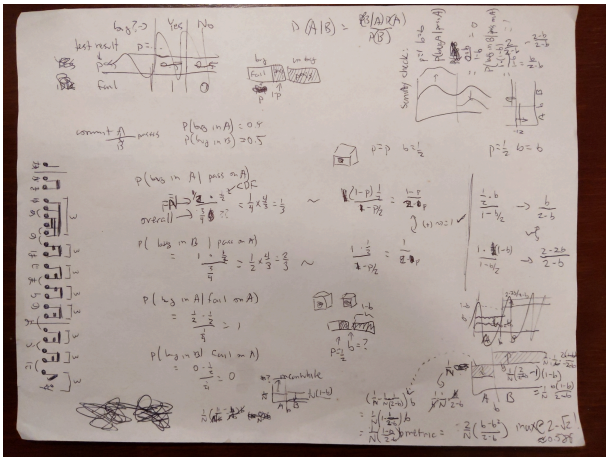


figure 2. derivation(?) of optimal(??) bisect point

5.2 bisect entropy

A PDF's information content is measured in entropy: $\mathcal{H}=\sum_i \mathsf{pdf}(i)\mathsf{ln}(\mathsf{pdf}(i))$. Alone, this value is fairly meaningless, but to compare them, a spikier PDF has lower entropy than a flatter one. Indeed, when the search terminates in Determinism World, $\mathcal{H}=0$. I thought for a while about how to link "minimum expected entropy" to the stated goal of $z=\mathsf{max}_i(\mathsf{pdf}(i))>0.99999$, but couldn't really formalize the idea. The goal of five 9's is fairly arbitrary anyway, and also not necessarily stable under the entropy measurement, since it doesn't care how the remaining 0.00001 is distributed.[8]

This strategy is expensive to compute. Whereas bisecting at some fixed threshold of probability mass merely requires a $O(\log n)$ query of the CDF, computing the expected entropy is $O(n)$ for *each* bisect point. A Very Smart Math Friend of mine (Gorey 2009) analyzed the easy case of the initial test, when the PDF is still uniform (i.e., $\forall i,\mathsf{cdf}(i)=i$), and found the closed form:

$$\mathcal{H}_{0,j}=p\ln(j)+(1-p\,j)(\ln(1-p\,j))-j(1-p)(\ln(j-p))$$

whose derivative, in his words, "looks like a giant mess that does not admit an analytic solution for j."

There is a ray of hope, however: thanks to the nondecreasing-PDF invariant I mentioned in section 3, the expected entropy has a unique local minimum.[9] Thus we can binary search for the global

[8] See `fn test_entropy_stability()`.

[9] See `fn test_expected_entropy_has_unique_local_minimum()`.

minimum, making this at worst $O(n \log n)$ instead of $O(n^2)$.

5.3 bisect randomly

Maybe a random world calls for a random algorithm! It's obvious this will terminate, but of course it won't be fast. Intuitively, if we use the PDF to weight the random choice, it will be faster than choosing uniformly in $[0, b)$. But how much faster?

You can tell at this point I'm starting to get statistics fatigue. Not unprecedented in these hallowed proceedings (Wise 2017).

5.4 human

Humans are known to be less patient than computers (citation: section 5.3). If a human is unwilling to compute $\mathcal{R} \circ \mathrm{pdf}$ by hand every step, or, more prohibitively, simply doesn't know p in advance so can't compute $\mathcal{R}$ at all, what should they do? Let's say a "human strategy" is any that doesn't use the PDF/CDF when deciding where to bisect.[10]

The most straightforward thing for a human to do is to assume $p = 1$ until they encounter evidence to the contrary. They'll conduct a normal binary search, and when they think they've found the bug at c_b, they'll just keep probing c_{b-1} until enough probability mass (invisible to them) moves to b and the simulator says they can stop. If this instead repros the bug at c_{b-1} after all, the human gets confused! They'll assume the bug might have been present as early as c_0, forgets their lower bound, and begins binary searching anew in the range $[0, b)$. It's easy to see this makes progress. Let's call this the "confused, forgetful human" strategy.

Another confused human variant is for them to remember their previous lower bounds. After all, just because their best lower bound $b-1$ was contradicted doesn't say anything about some earlier bound $a < b-1$ they might have observed. So this human will maintain a stack of lower bounds, and when $b-1$ is contradicted, they'll pop off the stack and resume searching in $[a, b)$ instead. Let's call this the "confused human who remembers". Incidentally, while implementing this strategy, I accidentally wrote the comment,

```
// the human, who now has perfect memory,
// walks backwards in time
```

and giggled a lot to myself as I imagined *sigbovik* fanfiction.

Finally, let's imagine the human knows in advance that maybe something is up with this p stuff. Wishing to carry on classically as though p were 1, they'll try to emulate p being higher by retrying any passing test "just to be sure", before moving on. If they retry the test r times, the probability it fails given the bug is present is then $1-(1-p)^{1+r}$. But it comes at the cost of r more steps for each new lower bound! As before, when this human thinks they're done, they'll repeatedly probe c_{b-1} until contradicted (and we'll make them forgetful, to keep things simple). Let's call this the "suspicious human of r retries".

6. simulated it

At this point I threw in the towel on the maths front, and wrote a bunch of code to let the law of large numbers do my work for me.

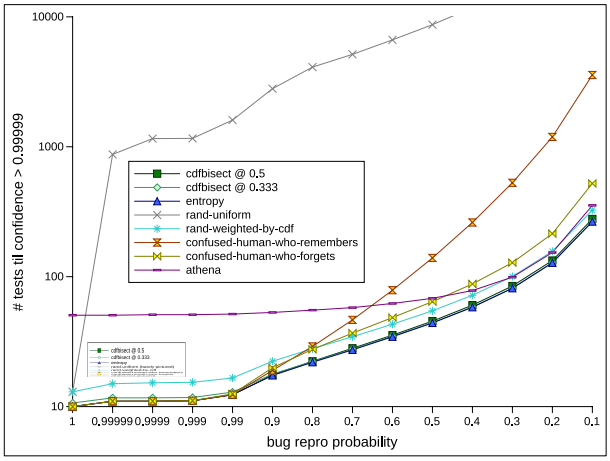
To save you some page-turning, here's the source code link again: `https://github.com/bblum/sigbovik/blob/master/bisect`. A `BisectStrategy` implements a function over p, the PDF, and/or its own internal state to choose the next bisect point. The `SimulationState` is initialized with fixed n and p, and repeatedly invokes a given `BisectStrategy`, hitting the PDF with Bayes's rule accordingly, until $z > 0.99999$. I provide implementations for all section 5's strategies in `src/strategies/`. It's written in Rust so it's self-documenting.

I learned a lot about floating point. I don't mean like NaNs and mantissa bits, I mean that when your project is fundamentally a long multiplication chain of increasingly high rationals, your precious $\mathrm{cdf}(n)$ inevitably drifts farther from 1 than you can bound with any arbitrary ϵ, and you start drowning in imprecision (Shiranu 2071). I suppose I could have used the `BigRational` library, but I chose Rust over Haskell for this so I could *avoid* runaway memory consumption... so, I resigned myself to explicitly renormalizing my PDFs by $1/\mathrm{cdf}(n)$ each time I updated them. After that, I was able to write some assertions to bound the imprecision drift within k(`std::f64::EPSILON`) (VII 2014). But yeah, I definitely spent some sanity points writing a function named `fn assert_kinda_equals_one(&self)`.

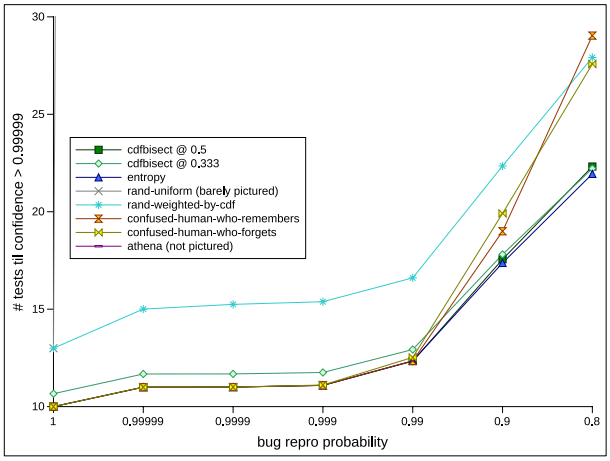
[10] In fact rand-uniform from last section counts as a human strategy. Did you bring your dice?

7. experiments

Let's get right to the good stuff. figure 3 shows the overall results, plotting the average performance of each of the strategies from section 5.



(a) full view



(b) detail view of high p range; linear scale

figure 3. perf results. how many bisect steps each strategy takes to reach five 9s of confidence

Each data point is simulated with $n = 1024$ and averaged over 65536 trials. I chose a power of two for n to minimize floating point drift (handwave), and a multiple of n for trials so that `buggy_commit = trial % n` would be uniform.

Of course, each trial takes exponentially longer and longer as $p \to 0$, so I didn't bother testing past 0.1. I also didn't bother measuring execution time, as prior work has shown that one's personal laptop is a fraught environment for stable performance evaluation (Blum 2018), but it was still very obvious that entropy was far, far slower per step than all other strategies. It was even slower than rand-uniform!

7.1 computer strategies

Minimizing expected entropy turned out to be the best strategy, globally across all p. Bisecting probability mass turns out to be preeetty close, although its performance varies depending on its bisect-point parameter j. Obviously, when $p = 1$, j should be 0.5 to reproduce classical binary search, but at around $p = 0.8$, $j = 0.333$ overtakes it. Ultimately at $p = 0.1$, the former is about 4% slower compared to the latter and to entropy, contradicting the "independent of p" hypothesis from section 5.1. I investigated a little more to see how p affected the optimal j, testing various js in increments of 0.01:

p	best j	cdfbisect(j)
1.0	0.50	10
0.9	0.45	17.4
0.8	0.41	21.9
0.7	0.39	27.5
0.6	0.40	34.6
0.5	0.36	44.1
0.4	0.35	58.3
0.3	0.34	81.6
0.2	0.36	127.9
0.1	0.35	266.6

65536 trials here wasn't really enough to get variance under control, but it definitely seems to converge towards 0.333 or maybe 0.35 or something as $p \to 0$. Open question why for warp zone, I guess.

rand-uniform is obviously terrible but I threw it in there for kicks. rand-cdf is not so terrible, doing even better than all the human strategies.

7.2 human strategies

I investigated how the suspicious human's number-of-retries parameter r affects their performance. In this notation, $r = 1$ means they retry once whenever the test passes, meaning two test runs on every passing commit. $r = 0$ is equivalent to the forgetful strategy, and $r = 9$ is what Athena does (section 4).

figure 3 was getting a little crowded so I've plotted all the human-friendly strategies separately on figure 4. It's pretty self-explanatory imo.

One thing that surprised me was that the human who forgets all their past known lower bounds when *any* of them is contradicted performed better than

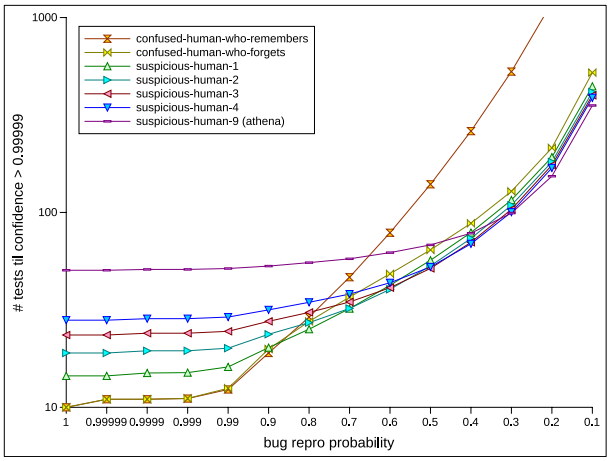


figure 4. version of figure 3 for human strategies

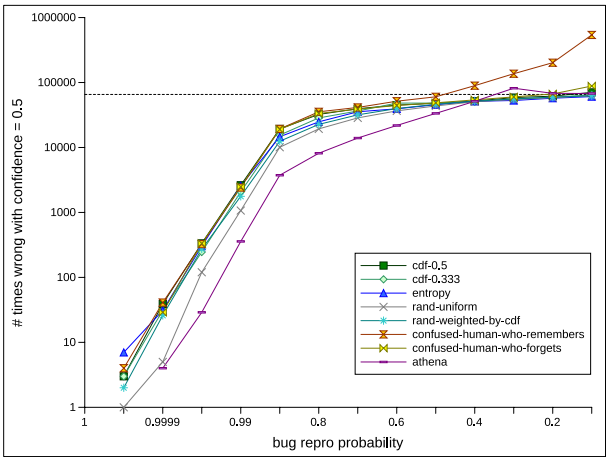


figure 5. how often each strategy was wrong, when terminating with only 50% confidence. the dashed line marks the approximate convergence point at 65536, equal to the number of right answers

the human who tried to rewind to their last not-yet-contradicted lower bound. I guess there's a moral here about not clinging to the past, moving on with life, or something.

7.3 why five 9s?

Let's talk about the arbitrary confidence threshold. There's a lot of knobs to tweak here and only so much time til April 1 (not to mention reader patience), but I did at least want to show what happens if you terminate the search when you're only 50% confident. Compared to figure 3, the strategies all terminate in about half as many steps, with the same relative ranking and break-even points. What's more interesting is the number of times they were wrong.

I had the simulator refuse to count a wrong result, restarting the search until it correctly identified the buggy commit 65536 times, and counting the number of restarts. With $z > 0.99999$ this was not very interesting: the strategies collectively made 0 errors 81% of the time, and were wrong more than twice only 0.6%. But $z > 0.5$ is a different story. As figure 5 shows, most strategies – even the comically slow rand-uniform – converged to being wrong about half the time, i.e., 65536 wrongs per 65536 rights. For higher p, athena was most resilient.

confused-human-who-remembers, alone, did not converge. They became more and more unable to cope with the repro rate approaching 0, utlimately being wrong 8 times as often as they were right. Oh, human... please learn to let go and love yourself for who you are. There's a better world out there, a sparkling radiant future waiting just for you!

8. future work

I'm not ashamed to admit I left a lot of loose ends in this paper. The biggest open questions for me are:

1. What relates p to cdfbisect's optimal argument?
2. Why is entropy *so tantalizingly close* to cdfbisect? Is there some way to show they're the same?

I have also prepared a set of backup questions in case those aren't enough for you:

4. If you don't know p in advance, is there an efficient way to multiplex the search with figuring it out on the fly?
5. What if false positives are also possible, with some flake rate $q \neq p$?
6. What if the cost of `git checkout` is nonzero (e.g., having to recompile your tests on new code)? Clearly human-mistrustful becomes better. What about some wacky hybrid cdfbisect-mistrustful?

7. conclusion

If you are a computer, you can do pretty well by bisecting the probability mass somewhere between 1/3 and 1/2. If you are a human, you should forget everything you know as soon as you see new evidence that contradicts your priors.

And remember to be patient and kind. Your test suite is just trying its best!

acknowledgments

Thanks to Jason Reed, Lynn Chordbug, and Sujay Jayakar for thinking about this problem with me.

referents

B. Blum. Transactional memory concurrency verification with Landslide. *sigbovik 12*, 2018.

K. E. Gorey. A categorical primer. *sigbovik 3*, 2009.

A. Pelc. Searching with known error probability. Theoretical Computer Science 63, 1989.

R. Rivest, A. Meyer, D. Kleitman, K. Winklmann, and J. Spencer. Coping with errors in binary search procedures. Journal of Computer and System Sciences 20, 1980.

U. Shah. Athena: Our automated build health management system. Dropbox.Tech blog, 2019.

N. Shiranu. IEEE 755: Drowning point numbers. *sigbovik 65*, 2071.

V. V. Vargomax. Generalized super mario bros. is NP-complete. *sigbovik 1*, 2007.

T. VII. What, if anything, is epsilon? *sigbovik 8*, 2014.

J. A. Wise. Batch normalization for improved DNN performance, my ass. *sigbovik 11*, 2017.