The Wrong Direction of Jensen's Inequality Is Algorithmically Right

Or Zamir ✉
Princeton University, NJ, USA

Abstract

Let $\mathcal{A}$ be an algorithm with expected running time e^X, conditioned on the value of some random variable X. We construct an algorithm $\mathcal{A}'$ with expected running time $O\left(e^{\mathsf{E}[X]}\right)$, that fully executes $\mathcal{A}$. In particular, an algorithm whose running time is a random variable T can be converted to one with expected running time $O\left(e^{\mathsf{E}[\ln T]}\right)$, which is never worse than $O(\mathsf{E}[T])$. No information about the distribution of X is required for the construction of $\mathcal{A}'$.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Algorithm design techniques; Theory of computation → Computational complexity and cryptography

Keywords and phrases algorithms, complexity, Jensen's inequality

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.107

Category Track A: Algorithms, Complexity and Games

Acknowledgements The author would like to thank Avi Wigderson for pointing out important references.

1 Introduction

Let $\mathcal{A}$ be a *Las Vegas*[1] randomized algorithm. Assume that conditioned on the value of some random variable X, the expected running time of $\mathcal{A}$ is e^X. By Jensen's inequality, $\mathsf{E}[e^X] \geq e^{\mathsf{E}[X]}$, and in fact $\mathcal{A}$'s expected running time might be much larger than $e^{\mathsf{E}[X]}$: Consider for example X that gets the value $\frac{1}{p}\mathsf{E}[X]$ with probability p and 0 otherwise, for any choice of $p > 0$; While the expectation of X is always $\mathsf{E}[X]$, the expectation of e^X is $p \cdot e^{\frac{1}{p}\mathsf{E}[X]}$ which can be arbitrarily large. We show that, surprisingly, any such $\mathcal{A}$ can be converted to a different Las-Vegas randomized algorithm $\mathcal{A}'$ that gives the same answer yet runs in expected time $O\left(e^{\mathsf{E}[X]}\right)$. Transforming $\mathcal{A}$ to $\mathcal{A}'$ does not require any assumption or knowledge about the distribution of X.

▶ **Theorem 1.** *There exists an algorithm T that receives as an input a randomized* Las Vegas *algorithm $\mathcal{A}$, and fully executes it. If the expected running time of $\mathcal{A}$ is e^X when conditioned on the value of some random variable X, then the expected running time of $T(A)$ is $O\left(e^{\mathsf{E}[X]}\right)$.*

As a corollary, any algorithm whose running time is a random variable T can be converted to one with expected running time $O\left(e^{\mathsf{E}[\ln T]}\right)$, which is never worse than $O(\mathsf{E}[T])$.

Recently we used the following simple version of Theorem 1 in a late revision of [10] to substantially simplify the analysis in the paper. The paper improves the running time of exact exponential-time algorithms for general Constraint Satisfaction Problems.

[1] A randomized algorithm is called *Las Vegas* if it always returns the correct answer, but its running time is a random variable.

▶ **Lemma 2** (from an up-to-date version of [10]). *Let $\mathcal{A}$ be an algorithm with expected running time 2^X conditioned on the value of a random variable X. There exists an algorithm $\mathcal{A}'$ that fully executes $\mathcal{A}$ and has an expected running time of $O\left(2^{\mathsf{E}[X]} \cdot \mathsf{E}[X]\right)$. Transforming $\mathcal{A}$ to $\mathcal{A}'$ requires knowing $\mathsf{E}[X]$.*

In this paper we focus on Theorem 1 itself, obtaining an optimal version of it.

We transform an algorithm $\mathcal{A}$ by using a sequence of *truncated evaluations*. A *truncated evaluation* of an algorithm $\mathcal{A}$ for t steps is the process of running algorithm $\mathcal{A}$ and aborting its run if it did not fully execute in the first t computational steps of its run. Each of the algorithms we present is thus a sequence of values $t_1, t_2, \ldots, t_i, \ldots$ which we use as thresholds for truncated evaluations of $\mathcal{A}$. We stop at the first time $\mathcal{A}$ is fully executed. These thresholds can be defined deterministically or be random variables. In the simpler algorithms we present, the thresholds depend on $\mathsf{E}[X]$ or even on the entire distribution X. For the proof of Theorem 1 the thresholds are completely independent of X and $\mathcal{A}$.

Truncated evaluations are frequently used in complexity theory (for example, see the proof of the time and space hierarchies in [2]). The first algorithmic use of such a sequence of truncated evaluations that we are aware of, is by Alt, Guibas, Mehlhorn, Karp and Wigderson [1]. They used it to convert *Las Vegas* randomized algorithms to *Monte Carlo* randomized algorithms, with success probability larger than what Markov's inequality gives. Luby, Sinclair and Zuckerman [5] then introduced a universal strategy for truncated evaluations. That is a sequence that is guaranteed to run in time $O(s \log s)$ if there exists any sequence that runs in time $O(s)$ for the same algorithm. Our contribution thus is two-fold: first, we prove the existence of good strategies in terms of $\mathsf{E}[X]$, and second, we show that these strategies can be explicitly constructed (i.e., without paying additional logarithmic factors). Not paying additional factors guarantees, due to Jensen's inequality, that our transformed algorithm is never worse than the original algorithm.

A natural use for such theorems is the regime of exponential-time algorithms. Consider the following toy example. As part of the classic algorithm of Schöning [9] it was shown that given an assignment α and a 3-SAT formula φ it is possible to test in $O\left(2^r\right)$ expected time whether φ has a satisfying assignment α_0 with $HAM\left(\alpha_0, \alpha\right) \leq r$, where $HAM(\cdot,\cdot)$ is the standard Hamming distance. This claim was also derandomized later [6]. We can use this primitive naively to obtain a non-trivial 3-SAT algorithm: Pick a random assignment α, and then run the above procedure of Schöning. Let X be the Hamming distance between α and a satisfying assignment α_0 of the input formula φ, this is a random variable. Conditioned on X, the expected running time of our algorithm is 2^X. The expected running time of our algorithm is thus $\mathsf{E}\left[2^X\right] = \sum_{r=0}^{n} \binom{n}{r} 2^{-n} 2^r = 2^{-n}(1+2)^n = \left(\frac{3}{2}\right)^n$. Using this paper's main Theorem, on the other hand, we can notice that $E[X] = \frac{n}{2}$ and thus we can convert the above algorithm in a black-box manner into one with expected running time $2^{\mathsf{E}[X]} = \left(\sqrt{2}\right)^n < \left(\frac{3}{2}\right)^n$. We note that Schöning already presented an algorithm using this procedure that is faster than both of the algorithms above.

Another similar example is the famous PPSZ algorithm for solving k-SAT, including its recent improvements, and generalizations for CSPs [7, 4, 3, 10]. In these algorithms, a randomly chosen permutation determines the number of input variables that we need to *guess* the values of. The expectation of this number of variables is then analyzed. The success probability or running time is exponential in this number. In the original PPSZ algorithm the analyzed quantity is the success probability and thus Jensen's inequality is applicable to bound this probability from below. In other variations (including [10]), the analyzed quantity is the running time and then Jensen's inequality is no longer applicable and either a more complicated analysis or the statement of this paper is necessary. Further discussion on possible applications and in particular possible implications for SAT algorithms appears in Section 3.

1.1 Preliminaries

We use standard notation throughout the paper. The notation $\ln x$ is used for the natural logarithm, and $\log x$ is used for the base two logarithm.

▶ **Definition 3** (Iterated functions). *Let $f : \mathbb{R} \to \mathbb{R}$ be a function. We define the iterated functions $f^{(k)} : \mathbb{R} \to \mathbb{R}$ recursively as follows. $f^{(0)}(x) := x$, and for any $k > 0$ we let $f^{(k)}(x) := f\left(f^{(k-1)}(x)\right)$.*

▶ **Definition 4** (Star functions). *Let $f : \mathbb{R} \to \mathbb{R}$ be a function. Assume f is strictly increasing and strictly shrinking[2] for all $x \geq x_0$. The star function of f, defined with respect to x_0 for every $x \geq x_0$, is*

$$f^{\star}(x) = \min\{k \mid f^{(k)}(x) \leq x_0\}.$$

The (general) Tower function $Tower_b(n,\, x) : \mathbb{N} \times \mathbb{R} \to \mathbb{R}$ is defined as $f^{(n)}(x)$ where $f(x) = b^x$. The standard Tower function $Tower : \mathbb{N} \to \mathbb{N}$ is defined as $Tower(n) = Tower_2(n, 1)$. The discrete inverse of the Tower function is $\log^{\star}$, defined with respect to $x_0 = 2$. That is, $\log^{\star} n$ is the smallest integer such that $Tower(\log^{\star} n) \geq n$.

2 Proof of Theorem 1

We begin by presenting a simple proof of Lemma 2.

Let $\mathcal{A}$ be an algorithm whose expected running time is e^X when we condition on the value of some non-negative random variable X. We observe, by Markov's inequality, that

$$\Pr\left(X > \mathsf{E}[X] + 1\right) \leq \frac{\mathsf{E}[X]}{\mathsf{E}[X]+1} = 1 - \frac{1}{\mathsf{E}[X]+1}.$$

Hence, consider the following algorithm.

Algorithm 1 Simple repetition.

Input: $\mathcal{A}$, $\mathsf{E}[X]$.
1: **repeat**
2: Run $\mathcal{A}$ for $2e^{\mathsf{E}[X]+1}$ computational steps.
3: **until** $\mathcal{A}$ completed a run.

▶ **Lemma 5.** *Algorithm 1 is expected to terminate in $O\left(e^{\mathsf{E}[X]} \cdot \mathsf{E}\left[X\right]\right)$ time.*

Proof. If $X \leq \mathsf{E}[X]+1$ then the expected running time of $\mathcal{A}$ is at most $e^{\mathsf{E}[X]+1}$, and hence by Markov's inequality a truncated evaluation of $\mathcal{A}$ for $2e^{\mathsf{E}[X]+1}$ steps concludes with probability at least $\frac{1}{2}$. By another application of Markov's inequality we got $\Pr\left(X \leq \mathsf{E}[X]+1\right) > \frac{1}{\mathsf{E}[X]+1}$. Hence, the expected number of iterations until the truncated evaluations concludes is at most $2\left(\mathsf{E}\left[X\right]+1\right)$. Each iteration takes $O\left(e^{\mathsf{E}[X]}\right)$ time. ◀

[2] That is, $f(x) < x$.

2.1 Optimal bound when the distribution of X is known

The bound given by Markov's inequality in

$$\Pr\left(X \geq \mathsf{E}[X]+1\right) \leq \frac{\mathsf{E}[X]}{\mathsf{E}[X]+1} = 1 - \frac{1}{\mathsf{E}[X]+1}$$

is attained *only* by the following distribution of X:

$$\Pr\left(X=k\right) := \begin{cases} \frac{1}{\mathsf{E}[X]+1} & k=0 \\ 1-\frac{1}{\mathsf{E}[X]+1} & k=\mathsf{E}[X]+1 \end{cases}.$$

In this distribution, on the other hand, the value of X is very small with a relatively high probability. In particular, in the case where $X < \mathsf{E}[X]+1$ we need to run $\mathcal{A}$ for much less than $e^{\mathsf{E}[X]+1}$ computational steps. Hence, it is sensible to hope that every distribution X has some threshold other than $\mathsf{E}[X]+1$ for which an algorithm similar to Algorithm 1 results in a better bound. Consider the following algorithm, which is a generalization of Algorithm 1 in which the threshold can be arbitrary.

Algorithm 2 Simple repetition with variable threshold.

Input: $\mathcal{A}$, t.
1: **repeat**
2: Run $\mathcal{A}$ for $2e^t$ computational steps.
3: **until** $\mathcal{A}$ completed a run.

▶ **Lemma 6.** *Let X be a non-negative random variable. There exists $t \in [0, \mathsf{E}[X]+1]$ such that $\frac{e^t}{\Pr(X<t)} \leq e^{\mathsf{E}[X]+1}$.*

Proof. Assume by contradiction that $\frac{e^t}{\Pr(X<t)} > e^{\mathsf{E}[X]+1}$ for every $t \in [0, \mathsf{E}[X]+1]$. Equivalently,

$$\Pr\left(X \geq t\right) = 1 - \Pr\left(X < t\right) > 1 - e^{t-(\mathsf{E}[X]+1)}.$$

Therefore,

$$\begin{aligned} \mathsf{E}[X] &= \int_0^\infty \Pr(X \geq t)\,\mathrm{d}t \geq \int_0^{\mathsf{E}[X]+1} \Pr(X \geq t)\,\mathrm{d}t \\ &> \int_0^{\mathsf{E}[X]+1} \left(1 - e^{t-(\mathsf{E}[X]+1)}\right)\mathrm{d}t \\ &= (\mathsf{E}[X]+1) - \left(1 - e^{-(\mathsf{E}[X]+1)}\right) = \mathsf{E}[X] + e^{-(\mathsf{E}[X]+1)} \\ &> \mathsf{E}[X], \end{aligned}$$

which is a contradiction. ◀

Lemma 6 implies the following.

▶ **Corollary 7.** *For every distribution X there exists a value of $t = t(X)$ for which Algorithm 2 runs in $O\left(e^{\mathsf{E}[X]}\right)$ time.*

We note that the additive constant $+1$ in the exponent in Lemma 6 is necessary. For a parameter E, consider the random variable X supported on $[0, E+1+\ln\left(1+e^{-(E+1)}\right)]$ and distributed with density $f(x) := e^{x-(E+1)}$; Its expectation is

$$\begin{aligned}\mathsf{E}[X] &= \int_0^{E+1+\ln\left(1+e^{-(E+1)}\right)} x f(x)\,\mathrm{d}x \\ &= \left((x-1)\,e^{x-(E+1)}\right)\Big|_0^{E+1+\ln\left(1+e^{-(E+1)}\right)} \\ &= \left(E+\ln\left(1+e^{-(E+1)}\right)\right)\cdot\left(1+e^{-(E+1)}\right)+e^{-(E+1)} \\ &= E+O\left(e^{-(E+1)}\right) = E+o(1),\end{aligned}$$

where the $o(1)$ term is vanishing when $E \to \infty$. On the other hand, for any $t \geq 0$ we have

$$\frac{e^t}{\Pr\left(X<t\right)} = \frac{e^t}{\min\left(1,\ e^{t-(E+1)}-e^{-(E+1)}\right)} > \frac{e^t}{e^{t-(E+1)}} = e^{E+1}.$$

2.2 Optimal algorithm when the distribution of X is unknown

If the only thing known about the distribution of X is its expectation $\mathsf{E}[X]$, then there is no fixed value of t for which Algorithm 2 is better than Algorithm 1. Fix a value of $\mathsf{E}[X]$ and a choice of t. If $t < \mathsf{E}[X]$ then with the constant distribution $X \equiv \mathsf{E}[X]$ Algorithm 2 never terminates. Otherwise, $t \geq \mathsf{E}[X]$ and we consider the following distribution X:

$$\Pr\left(X=k\right) := \begin{cases} 1-\frac{\mathsf{E}[X]}{t+1} & k=0 \\ \frac{\mathsf{E}[X]}{t+1} & k=t+1 \end{cases}.$$

For this distribution, the expected running time of Algorithm 2 is

$$\frac{e^t}{1-\frac{\mathsf{E}[X]}{t+1}} = e^{\mathsf{E}[X]}\cdot\frac{e^s}{\left(\frac{s+1}{\mathsf{E}[X]+s+1}\right)} = e^{\mathsf{E}[X]}\left(1+\frac{\mathsf{E}[X]}{s+1}\right)e^s \geq e^{\mathsf{E}[X]}\mathsf{E}[X]\cdot\frac{e^s}{s+1} \geq e^{\mathsf{E}[X]}\mathsf{E}[X],$$

where $s := t-\mathsf{E}[X] \geq 0$ and the last inequality follows as $e^s \geq s+1$ for any s.

To improve Algorithm 1 then, we need to consider *several thresholds*. We demonstrate this idea with the following Lemma.

▶ **Lemma 8.** *Let X be a non-negative random variable. It holds that either $\Pr\left(X \leq \mathsf{E}\left[X\right]-\ln\mathsf{E}\left[X\right]\right) > \frac{1}{\mathsf{E}[X]+1}$ or $\Pr\left(X \leq \mathsf{E}\left[X\right]+2\right) > \frac{1}{\ln\mathsf{E}[X]+2}$.*

Proof. Assume that $p := \Pr\left(X \leq \mathsf{E}\left[X\right]-\ln\mathsf{E}\left[X\right]\right) \leq \frac{1}{\mathsf{E}[X]+1}$. We observe that

$$\begin{aligned}\mathsf{E}\left[X\right] &= p\,\mathsf{E}\left[X \mid X \leq \mathsf{E}\left[X\right]-\ln\mathsf{E}\left[X\right]\right] + (1-p)\,\mathsf{E}\left[X \mid X > \mathsf{E}\left[X\right]-\ln\mathsf{E}\left[X\right]\right] \\ &\geq (1-p)\,\mathsf{E}\left[X \mid X > \mathsf{E}\left[X\right]-\ln\mathsf{E}\left[X\right]\right],\end{aligned}$$

and hence

$$\begin{aligned}\mathsf{E}\left[X \mid X > \mathsf{E}\left[X\right]-\ln\mathsf{E}\left[X\right]\right] &\leq \frac{1}{1-p}\mathsf{E}[X] \\ &\leq \frac{1}{1-\frac{1}{\mathsf{E}[X]+1}}\mathsf{E}[X] \\ &= \mathsf{E}[X]+1.\end{aligned}$$

Denote by $Y := X - (\mathsf{E}[X] - \ln \mathsf{E}[X])$. The above can now be rephrased as $\mathsf{E}[Y \mid Y > 0] \leq \ln \mathsf{E}[X] + 1$. Applying Markov's inequality to Y conditioned on $Y > 0$, we get

$$\Pr(Y > \ln \mathsf{E}[X] + 2 \mid Y > 0) \leq \frac{\ln \mathsf{E}[X] + 1}{\ln \mathsf{E}[X] + 2} = 1 - \frac{1}{\ln \mathsf{E}[X] + 2}.$$

We conclude by noting that

$$\Pr(X > \mathsf{E}[X] + 2) = \Pr(Y > \ln \mathsf{E}[X] + 2) \leq \Pr(Y > \ln \mathsf{E}[X] + 2 \mid Y > 0). \quad \blacktriangleleft$$

Consider the following Algorithm.

Algorithm 3 Two thresholds algorithm.

Input: $\mathcal{A}$, $\mathsf{E}[X]$.
1: **repeat**
2: **for** $\lceil \mathsf{E}[X] + 1 \rceil$ **times do**
3: Run $\mathcal{A}$ for $2e^{\mathsf{E}[X] - \ln \mathsf{E}[X]}$ computational steps.
4: **for** $\lceil \ln \mathsf{E}[X] + 2 \rceil$ **times do**
5: Run $\mathcal{A}$ for $2e^{\mathsf{E}[X]+2}$ computational steps.
6: **until** $\mathcal{A}$ completed a run.

Due to Lemma 8, each iteration of the outermost loop of Algorithm 3 succeeds to fully execute $\mathcal{A}$ with probability larger than $1 - e^{-1}$. Thus, in expectation we run this loop for a constant number of iterations. The first **for** loop takes $O\left(\mathsf{E}[X] \cdot e^{\mathsf{E}[X] - \ln \mathsf{E}[X]}\right) = O\left(e^{\mathsf{E}[X]}\right)$ expected time, and the second takes $O\left(e^{\mathsf{E}[X]} \ln \mathsf{E}[X]\right)$. We conclude the following.

▶ **Corollary 9.** *Algorithm 3 runs in expected time* $O\left(e^{\mathsf{E}[X]} \ln \mathsf{E}[X]\right)$.

Intuitively, the *proof* of Lemma 8 can be viewed as a reduction from the variable X to the variable $Y \mid (Y > 0)$, that has a much lower expectation: $\mathsf{E}[Y \mid Y > 0] \leq \ln \mathsf{E}[X] + 1$. We can thus hope that iterating the proof for $\ln^\star \mathsf{E}[X]$ times would result in reducing X to a variable with constant expectation. A natural implementation of this idea would result in an algorithm that runs in expected time $O\left(e^{\mathsf{E}[X]} \ln^\star \mathsf{E}[X]\right)$. We next formalize this intuition, and do so in a more careful manner to avoid the $\ln^\star \mathsf{E}[X]$ factor.

▶ **Definition 10.** *Let* $\lambda(x) := 3\ln(x)$ *and note it is strictly increasing and shrinking for all* $x \geq 5$. *We define* $\lambda^\star(x)$, *for* $x \geq 5$, *to be the smallest* $k \in \mathbb{N}$ *such that* $\lambda^{(k)}(x) \leq 5$.

▷ Claim 11. The following hold for all $x \geq 5$:
1. $\lambda^\star(x) = \Theta(\log^\star x)$.
2. $\lambda^{(\lambda^\star(x))}(x) > 4$.
3. $\sum_{i=0}^{\lambda^\star(x)} \frac{1}{\lambda^{(i)}(x)} < 2$.

Proof. (1) We have that $\lambda^{(2)}(x) \leq \log x \leq \lambda(x)$ for all $x \geq 410$. In particular, $\log^\star x \leq \lambda^\star(x) \leq 2\log^\star x + \lambda^\star(410)$.

(2) $\lambda^{(\lambda^\star(x)-1)}(x) > 5$ and hence $\lambda^{(\lambda^\star(x))}(x) > \lambda(5) > 4.82$.

(3) For all $x \geq 17$ it holds that $\lambda(x) \leq \frac{x}{2}$. Let k' be the smallest integer such that $\lambda^{(k')}(x) < 17$. We thus have

$$\sum_{i=0}^{k'-1} \frac{1}{\lambda^{(i)}(x)} < \frac{1}{17} \sum_{i=0}^{\infty} 2^{-i} = \frac{2}{17}.$$

On the other hand, there are at most $\lambda^\star(17)$ summands that are strictly larger than $\frac{1}{17}$, thus by (2) we have

$$\sum_{i=k'}^{\lambda^\star(x)} \frac{1}{\lambda^{(i)}(x)} < \frac{\lambda^\star(17)}{4} = \frac{5}{4}. \qquad \lhd$$

We are now ready to prove a generalized version of Lemma 8, that is going to be the core of our final algorithm.

▶ **Lemma 12.** *Let X be a non-negative distribution and $E \geq \max(\mathsf{E}[X], 5)$ be an upper bound on its expectation. There either exists $1 \leq k \leq \lambda^\star(E)$ such that $\Pr\left(X < E - \lambda^{(k)}(E)\right) \geq \left(\left(\lambda^{(k-1)}(E)+2\right)^2+1\right)^{-1}$, or it holds that $\Pr(X < E+10) \geq \frac{1}{2}$.*

Proof. We recursively denote by $E_0 := E$ and by $E_k := \lambda^{(k)}(E) + \sum_{i=0}^{k-1} \frac{1}{E_i}$ for $1 \leq k \leq \lambda^\star(E)$. Note that $E_k \geq \lambda^{(k)}(E)$ and hence also

$$E_k = \lambda^{(k)}(E) + \sum_{i=0}^{k-1} \frac{1}{E_i} \leq \lambda^{(k)}(E) + \sum_{i=0}^{k-1} \frac{1}{\lambda^{(i)}(E)} < \lambda^{(k)}(E) + 2,$$

where the last inequality follows from Claim 11. In particular, $\lambda^{(k)}(E) \leq E_k < \lambda^{(k)}(E) + 2$.

Assume that $\Pr\left(X < E - \lambda^{(k)}(E)\right) < \left(\left(\lambda^{(k-1)}(E)+2\right)^2+1\right)^{-1} < \frac{1}{(E_{k-1})^2+1}$ for every $1 \leq k \leq \lambda^\star(E)$.

Denote by $Y_k := X - \left(E - \lambda^{(k)}(E)\right)$ for $k \geq 0$. We prove by induction on k that $\mathsf{E}[Y_k \mid Y_k \geq 0] \leq E_k$. For $k = 0$ the claim is straightforward as $Y_0 = X$ and $E_0 = E$. For the inductive step, we assume the hypothesis holds for $k-1$ and show it holds for k. We note that $Y_{k-1} > Y_k$ and hence if $Y_k \geq 0$ then $Y_{k-1} \geq 0$ as well. Hence,

$$\begin{aligned}\mathsf{E}[Y_{k-1} \mid Y_{k-1} \geq 0] &\geq \Pr(Y_k \geq 0 \mid Y_{k-1} \geq 0)\, \mathsf{E}[Y_{k-1} \mid Y_k \geq 0] \\ &\geq \Pr(Y_k \geq 0)\, \mathsf{E}[Y_{k-1} \mid Y_k \geq 0].\end{aligned}$$

Thus, by the induction hypothesis we have

$$\begin{aligned}\mathsf{E}[Y_{k-1} \mid Y_k \geq 0] &\leq \frac{\mathsf{E}[Y_{k-1} \mid Y_{k-1} \geq 0]}{\Pr(Y_k \geq 0)} \\ &\leq \frac{E_{k-1}}{1 - \frac{1}{(E_{k-1})^2+1}} \\ &= E_{k-1} + \frac{1}{E_{k-1}}.\end{aligned}$$

Therefore,

$$\begin{aligned}\mathsf{E}[Y_k \mid Y_k \geq 0] &= \mathsf{E}[Y_{k-1} \mid Y_k \geq 0] + \lambda^{(k)}(E) - \lambda^{(k-1)}(E) \\ &\leq E_{k-1} + \frac{1}{E_{k-1}} + \lambda^{(k)}(E) - \lambda^{(k-1)}(E) \\ &= E_k.\end{aligned}$$

In particular, we have that $\mathsf{E}\left[Y_{\lambda^\star(E)} \mid Y_{\lambda^\star(E)} \geq 0\right] \leq E_{\lambda^\star(E)} < \lambda^{(\lambda^\star(E))}(E) + 2 \leq 7$.

Therefore,

$$\begin{aligned}\Pr\left(X \geq E+10\right) &\leq \Pr\left(X \geq E+10 \;\middle|\; X \geq E-\lambda^{(\lambda^\star(E))}(E)\right)\\ &= \Pr\left(Y_{\lambda^\star(E)} \geq \lambda^{(\lambda^\star(E))}(E)+10 \;\middle|\; Y_{\lambda^\star(E)} \geq 0\right)\\ &\leq \Pr\left(Y_{\lambda^\star(E)} \geq 14 \;\middle|\; Y_{\lambda^\star(E)} \geq 0\right) < \frac{7}{14} = \frac{1}{2}.\end{aligned}$$

◀

Consider the following algorithm.

Algorithm 4 Multiple thresholds algorithm.

Input: $\mathcal{A}$, E.

1: **repeat**
2: **for** $k = 1$ to $\lambda^\star(E)$ **do**
3: **for** $2\lceil\left(\lambda^{(k-1)}(E)+2\right)^2+1\rceil$ **times do**
4: Run $\mathcal{A}$ for $2e^{E-\lambda^{(k)}(E)}$ computational steps.
5: **for** 2 **times do**
6: Run $\mathcal{A}$ for $2e^{E+10}$ computational steps.
7: **until** $\mathcal{A}$ completed a run.

▶ **Corollary 13** (of Lemma 12). *Each* ***repeat*** *loop of Algorithm 4 fully executes $\mathcal{A}$ with probability at least $\frac{3}{4}$.*

▶ **Lemma 14.** *Let $E \geq \max(\mathsf{E}[X],\ 5)$, Algorithm 4 runs in $O\left(e^E\right)$ expected time.*

Proof. By Corollary 13 we enter the **repeat** loop a constant number of times in expectation. We thus analyze the computational cost of a single **repeat** loop. The evaluations in Lines $5-6$ take $O\left(e^E\right)$ time. The evaluations in Lines $2-4$ take

$$\sum_{k=1}^{\lambda^\star(E)} 2\lceil\left(\lambda^{(k-1)}(E)+2\right)^2+1\rceil \cdot 2e^{E-\lambda^{(k)}(E)} = O\left(e^E \cdot \sum_{k=1}^{\lambda^\star(E)} \left(\lambda^{(k-1)}(E)\right)^2 e^{-\lambda^{(k)}(E)}\right)$$

time. By the definition of $\lambda(x)$, we have $e^{-\lambda^{(k)}(x)} = e^{-3\ln\left(\lambda^{(k-1)}(x)\right)} = \left(\lambda^{(k-1)}(x)\right)^{-3}$. In particular,

$$\sum_{k=1}^{\lambda^\star(E)} \left(\lambda^{(k-1)}(E)\right)^2 e^{-\lambda^{(k)}(E)} = \sum_{k=1}^{\lambda^\star(E)} \left(\lambda^{(k-1)}(E)\right)^{-1} < 2,$$

where the last inequality follows from Claim 11. ◀

Finally, we also get rid of the necessity to provide the algorithm with E or $\mathsf{E}[X]$.

▶ **Theorem 15.** *Algorithm 5 runs in expected time $O\left(e^{\mathsf{E}[X]}\right)$.*

Proof. By Lemma 14 the iteration of the outermost **for** loop corresponding to E takes at most $C \cdot e^E$ time, for some global constant C. All iterations in which $E < \mathsf{E}[X]$ thus take $O\left(e^{\mathsf{E}[X]}\right)$ time. By Corollary 13, each subsequent iteration succeeds with probability at least $\frac{3}{4}$. Thus the expected running time is bounded by

$$Ce^{\mathsf{E}[X]} \cdot \sum_{t=0}^{\infty} e^t \left(\frac{1}{4}\right)^t = O\left(e^{\mathsf{E}[X]}\right).$$

◀

Algorithm 5 Final algorithm.

Input: $\mathcal{A}$.

for $E = 5$ to ∞ **do**
  for $k = 1$ to $\lambda^\star(E)$ **do**
    for $2\lceil\left(\lambda^{(k-1)}(E)+2\right)^2+1\rceil$ **times do**
      Run $\mathcal{A}$ for $2e^{E-\lambda^{(k)}(E)}$ computational steps.
  for 2 **times do**
    Run $\mathcal{A}$ for $2e^{E+10}$ computational steps.
  return if $\mathcal{A}$ completed a run.

3 Conclusions and Open Problems

We showed that a Las-Vegas algorithm with expected running e^X conditioned on the value of some random variable X, can always be converted to a Las-Vegas algorithm with expected running time $O\left(e^{\mathsf{E}[X]}\right)$. In particular, an algorithm whose running time is a random variable T can be converted to one with expected running time $O\left(e^{\mathsf{E}[\ln T]}\right)$, which is never worse than $O(\mathsf{E}[T])$.

We demonstrated a use of this theorem to simplify a proof in the regime of exponential time algorithms [10]. It is interesting to try applying it to other exponential and non-exponential time algorithms and see if it can simplify or even improve the analysis.

3.1 Considering the variance

In terms of $\mathsf{E}[X]$ only, we can not get any better than $O\left(e^{\mathsf{E}[X]}\right)$ as the distribution of X might be constant. In that case though, the variance of X is zero. Can we get a better bound just by assuming that the variance of X is large? Unfortunately, with the standard definition of variance this is not the case. For any choice of E and $V \geq 2E^2e^{-E}$ consider the following distribution:

$$\Pr(X=k) := \begin{cases} e^{-E} & k=0 \\ 1-\frac{Ve^{-E}}{V-E^2e^{-E}} & k=E \\ \frac{\left(Ee^{-E}\right)^2}{V-E^2e^{-E}} & k=\frac{V}{Ee^{-E}} \end{cases}.$$

Its expectation is E, its variance is V, which can be arbitrarily large, and nevertheless $\Pr(X < E) = e^{-E}$ so no strategy can beat $O\left(e^E\right)$.

On the other hand, the wishful thinking above is true with some other notions of deviation. For example, if we consider mean absolute deviation instead of standard deviation (i.e., $\mathsf{E}\left[\,|X-\mathsf{E}[X]|\,\right]$), then *it is* true that if the deviation is large then we can get a better running time. It is intriguing to find useful notion of deviation for which such a statement is true, with the goal of improving the running time of algorithms by analyzing the deviation of X.

In particular, consider the PPSZ algorithm for solving k-SAT [7] [4]. The algorithm uses randomization in two ways: first, a random permutation of the variables in the input formulas is drawn; Then, the chosen permutation determines the number of variables we need to *guess* the value of. In a recent improvement of the PPSZ analysis, Scheder [8] showed that in some large subset of permutations the number of guessed variables is smaller than what we expect when taking a uniformly random permutation. In particular, this implies that there is some non-negligible variance in the original algorithm's running time. Can we get better SAT algorithms by analyzing this variance?

References

1 Helmut Alt, Leonidas Guibas, Kurt Mehlhorn, Richard Karp, and Avi Wigderson. A method for obtaining randomized algorithms with small tail probabilities. *Algorithmica*, 16(4):543–547, 1996.

2 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

3 Thomas Dueholm Hansen, Haim Kaplan, Or Zamir, and Uri Zwick. Faster k-sat algorithms using biased-ppsz. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 578–589, 2019.

4 Timon Hertli. 3-sat faster and simpler—unique-sat bounds for ppsz hold in general. *SIAM Journal on Computing*, 43(2):718–729, 2014.

5 Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.

6 Robin A Moser and Dominik Scheder. A full derandomization of schöning's k-sat algorithm. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 245–252, 2011.

7 Ramamohan Paturi, Pavel Pudlák, Michael E Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. *Journal of the ACM (JACM)*, 52(3):337–364, 2005.

8 Dominik Scheder. Ppsz is better than you think. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 205–216. IEEE, 2022.

9 T Schoning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 410–414. IEEE, 1999.

10 Or Zamir. Faster algorithm for unique $(k, 2)$-csp. *ESA*, 2022.