The LBRY Trending Algorithm

Brendon J. Brewer

Abstract

I describe the most recent update to the LBRY trending algorithm. The ideas are similar to previous implementations, but with a few new features.

1 Introduction

LBRY [Li et al., 2020] needs a trending algorithm to rank claims by 'recent popularity', preferably using on-chain information such as changes in support. An intuitive idea is to have increases and decreases in overall support cause a 'spike' in the trending score, which then fades away over time (assuming no further changes occur). This idea has formed the basis of all the implementations that I have written.

All claims, be they streams/files, channels, reposts, or collections, have an associated trending score. Hubs (formerly wallet servers) compute and save the trending score of every claim along with the other data. Several views of content on LBRY Desktop and Odysee involve sorting claims by their trending score, in descending order.

In 2021, Jack Robison rewrote a lot of the hub code to improve performance and to ensure hub data integrity when a chain reorganisation occurs. The backend now uses leveldb to store transaction and claim information, and Elasticsearch to store claim information in a suitable form for clients to query. Leveldb will be replaced with rocksdb soon. As part of these changes, trending needed to be made more efficient. This meant we made some modifications to it, some of which are pretty neat in my opinion.

2 Exponentially Decaying Spikes

Suppose that, at time (block height) t, the total amount of LBC staked towards a claim is ℓ_t, and its trending score is Y_t. At height $(t+1)$, if the total amount of LBC remains the same, the trending score is decayed towards zero:

$$Y_{t+1} = kY_t, \tag{1}$$

where k is a decay constant which we set equal to $\exp(-1/576) \approx 0.9982653$, so that the e-folding timescale is 576 blocks, approximately 24 hours.

If an LBC change occurs, the update is instead

$$Y_{t+1} = kY_t + M(\ell_t, \ell_{t+1}), \tag{2}$$

where the second term is the 'mass' of a 'trending spike', and depends on the old and new LBC values. A simple idea is to set the mass of the trending spike equal to the change in LBC:

$$M(\ell_t, \ell_{t+1}) = \ell_{t+1} - \ell_t. \tag{3}$$

However, in practice this simple choice might lead to undesirable results. Many supports on the LBRY network are created by publishers themselves, attempting to boost their own content's visibility. If an item receives many organic views and a moderate number of small tips, one might expect it to trend, yet with the linear choice for M, it will have no hope of competing with a 'whale' who can deposit 100,000 LBC in support without blinking an eye. In Section 3, we simulate several different patterns of support, and choose a function M such that the size of trending spikes makes 'minnows' able to compete with 'whales', at least to a greater extent than they can with the linear choice.

3 Nonlinearity — the Spike Mass Function

As briefly discussed in the last section, a linear choice of spike mass function might not be ideal. When we choose this function, we are making a decision about how significant one amount of LBC at one time is, compared to a different amount of LBC at a different time. For example, perhaps 1000 LBC right now should be considered as significant as 10,000 LBC 12 hours ago.

3.1 Change in Softened LBC Amount

Our final choice of spike mass function is complex. I will describe simpler versions at first, which will help in motivating the final, complicated, choice. First, let's look at a choice where the LBC amounts are 'softened' by raising them to some power less than one (here I have chosen a power of 1/3, in previous versions of the algorithm I used 1/4):

$$M_1(\ell_t, \ell_{t+1}) = \ell_{t+1}^{1/3} - \ell_t^{1/3}. \tag{4}$$

The size of a trending spike here would be the change in the 'softened' LBC amount. A change from 0 to 1 LBC would give a spike mass of 1, as would a change from 1 to 1000 LBC, or 1000 to 1,000,000. It takes more LBC to move a mountain than to move a molehill.

With this choice, it wouldn't matter whether a change took place in one large step or in smaller increments[1]. Specifically,

$$\begin{aligned} M_1(x,y) + M_1(y,z) &= y^{1/3} - x^{1/3} + z^{1/3} - y^{1/3} && (5)\\ &= z^{1/3} - x^{1/3} && (6)\\ &= M_1(x,z). && (7) \end{aligned}$$

This may not be entirely desirable. Genuinely popular claims on LBRY tend to receive several smaller supports, rather than one large one, and intuitively this ought to help. A trending algorithm should show what's trending, as well as what's recently been boosted.

3.2 Softened Change in LBC Amount

Instead of applying the power before the subtraction, we could apply the subtraction first — i.e., we could use

$$M_2(\ell_t, \ell_{t+1}) = (\ell_{t+1} - \ell_t)^{1/3}. \tag{8}$$

The subscript 2 distinguishes this choice from $M_1()$ in the previous subsection. This would make several small supports more effective than one large support, in terms of the trending score, a

[1]This would occur with any choice of the form $M(x,y) = \Phi(y) - \Phi(x)$ for a monotonic function $\Phi()$.

potentially desirable feature:

$$M_2(x,y) + M_2(y,z) = (y-x)^{1/3} + (z-y)^{1/3} \tag{9}$$
$$> (z-x)^{1/3}. \tag{10}$$

The inequality follows from (I don't know — Jensen's inequality perhaps). However, in numerical experiments, we found that this effect was too strong, and whales could attain huge heights in trending score by splitting up, say, 100,000 LBC, into several supports.

3.3 Interpolation

The two ideas $M_1()$ and $M_2()$ can be merged into a compromise third option, as follows:

$$M_3(\ell_t, \ell_{t+1}) = M_1(\ell_t, \ell_{t+1})^{\alpha} M_2(\ell_t, \ell_{t+1})^{1-\alpha}. \tag{11}$$

This is linear interpolation of $\log M$ values, or alternatively, it is 'geometric interpolation'. If α is close to 1 then the rule is more like M_1, if it is close to zero it is more like M_2.

What we actually settled on was to have $\alpha \approx 0.5$ when we're dealing with low LBC values, and $\alpha = 0.85$ when dealing with higher LBC values. There is a narrow range over which α is between these two values:

$$\alpha = \begin{cases} 0.5, & \ell_2 \leq 50, \\ 0.01\ell_2, & 50 < \ell_2 \leq 85, \\ 0.85, & \text{otherwise.} \end{cases} \tag{12}$$

This completes the specification of the spike mass function.

4 Sparsity

For most claims, most of the time, nothing happens. However, computing the exponential decay using Equation 1 would still incur a cost. Right now, there are about 15 million active claims on the LBRY network, and the intention is that there might be orders of magnitude more in the future. We want to avoid decaying the trending score of 15 million claims when nothing has happened to most of them. In the most recent implementation, we have devised a way to avoid decaying any claims, ever — only touched claims (those whose LBC values changed) ever need any computation.

The formula for updating a trending score is Equation 2:

$$Y_{t+1} = kY_t + M(\ell_t, \ell_{t+1}). \tag{13}$$

However, instead of storing Y for each claim, we could instead store a transformed version of it:

$$Z_t = Y_t k^{-t}. \tag{14}$$

The updating formula for Y can then be rewritten as

$$Z_{t+1}k^{t+1} = kZ_t k^t + M(\ell_t, \ell_{t+1}). \tag{15}$$

Solving for Z_{t+1} gives

$$Z_{t+1} = kZ_t k^t k^{-(t+1)} + M(\ell_t, \ell_{t+1})k^{-(t+1)} \tag{16}$$
$$= Z_t + M(\ell_t, \ell_{t+1})k^{-(t+1)}. \tag{17}$$

Thus, if we maintain Z instead of Y, we replace 'exponential decay' with maintenance of a constant trending score for all untouched claims — the formula reduces to $Z_{t+1} = Z_t$. As time goes on, the mass of applied spikes merely grows exponentially (recall that k is just below 1, so k raised to a negative power becomes large and positive), and that is how recent changes become more significant than old ones.

The only difficulty now is how to add exponentially-large values to touched claims while avoiding overflow. This requires another nonlinear transformation which I informally call 'squashing'.

5 The 'Squashed' Grade

Suppose x is a huge number, where double precision overflow might occur if we were to use it in a calculation (this happens at around 10^{308}). If x is positive, we could instead deal with logarithms. Then, to add two huge numbers without overflow, we could use

$$\begin{aligned} \log(x+y) &= \log(\exp(\log x) + \exp(\log y)) && (18)\\ &= \text{logsumexp}(\log x, \log y), && (19) \end{aligned}$$

where logsumexp() is given by

$$\text{logsumexp}(a,b) = \max(a,b) + \log\left[\exp(a - \max(a,b)) + \exp(b - \max(a,b))\right]. \quad (20)$$

Subtracting the maximum before doing the exponentiation prevents overflow. Similarly, differences could be computed on the log scale. Assuming $x > y$, we can use the following method:

$$\begin{aligned} \log(x-y) &= \log(\exp(\log x) - \exp(\log y)) && (21)\\ &= \text{logdiffexp}(\log x, \log y), && (22) \end{aligned}$$

where logdiffexp is given by

$$\text{logdiffexp}(a,b) = a + \log\left[1 - \exp(b-a)\right] \quad (23)$$

for $a > b$.

However, logarithms only work for positive x, and our trending scores can be negative (and zero) as well. In principle, any monotonic function will work (and will allow the same total ordering of claims), but we need one where arithmetic is possible without overflow.

Instead of log, define a squashing function S as:

$$S(x) = \begin{cases} \log(1+x), & \text{if } x \geq 0,\\ -\log(1-x), & \text{otherwise.} \end{cases} \quad (24)$$

The inverse, unsquashing function, is

$$U(x) = S^{-1}(x) = \begin{cases} \exp(x) - 1, & \text{if } x \geq 0,\\ 1 - \exp(-x), & \text{otherwise.} \end{cases} \quad (25)$$

For large positive values, these are approximately equal to log and exp respectively, but for smaller values (and negative values), it remains continuous.

To compute updated trending scores, we need to add and multiply values using Equations 17 and 11, but on the squashed grade. In other words, if we have squashed values x and y, we will need the squashed sum

$$x \oplus y = S(U(x) + U(y)) \quad (26)$$

and the squashed product

$$x \otimes y = S(U(x)U(y)). \tag{27}$$

These can be worked out in terms of logsumexp and logdiffexp. The results are fairly complicated and are given in the Appendix.

In the actual code, we store squashed trending values, i.e., $S(Z)$ in the notation of this paper, for every claim. When claims are touched, the squashed trending values are updated. At block height 1,000,000, which occurred in 2021, unsquashed trending values would have been of order

$$\exp\left(\frac{10^6}{576}\right) \approx 10^{753}. \tag{28}$$

However, the squashed values are only of order $10^6/576 \approx 1736$. Differences in squashed trending values, which determine the ordering of claims in applications, occur in the third, fourth, and subsequent digits. With double precision storage, the situation will remain acceptable for centuries. For example, at block height 100 million, expected around 2496 AD, squashed trending scores will be around 150,000, with plenty of remaining significant digits to allow for sorting.

6 Acknowledgements

BJB would like to thank Kevin Knuth (SUNY Albany) for inspiration.

References

Jun Li, Alex Grintsvayg, Jeremy Kauffman, and Charles Fleming. Lbry: A blockchain-based decentralized digital content marketplace. In *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 42–51. IEEE, 2020.

A Squashed Arithmetic

The squashed addition operator is broken down into four cases, depending on the sign of the arguments:

$$x \oplus y = \begin{cases} -\text{logsumexp}(-x, \text{logdiffexp}(-y, 0)), & x < 0 \text{ and } y < 0, \\ \text{logsumexp}(x, \text{logdiffexp}(y, 0)), & x \geq 0 \text{ and } y \geq 0 \\ \text{logsumexp}(0, \text{logdiffexp}(x, -y)), & \text{sign}(x) \neq \text{sign}(y) \text{ and } |x| \geq |y| \\ -\text{logsumexp}(0, \text{logdiffexp}(-y, x)), & \text{sign}(x) \neq \text{sign}(y) \text{ and } |x| < |y| \end{cases}. \tag{29}$$

Squashed multiplication is simpler:

$$x \otimes y = \text{sign}(xy)\text{logsumexp}\left(\text{logdiffexp}(|x|, 0) + \text{logdiffexp}(|y|, 0), 0\right). \tag{30}$$