Dipper Proof-of-work

@Datavetaren

January 11, 2019

Abstract

Proof-of-work is the mechanism to prove that work has been done. It typically amounts to a ("random") search problem where once a solution is found, it is easy (and requires almost no resources) to verify. Traditionally, these proof-of-work algorithms have solely been focused on CPU power, but new algorithms have emerged that take advantage of other resources such as memory. As the value of cryptocurrencies go up there has been a race among vendors to provide application specific circuits (ASICs) which bring an advantage over traditional off-the-shelf hardware for that particular proof-of-work problem. This comes with pros & cons: specialized hardware dedicated to a certain cryptocurrency secures its network, but as specialized hardware is a complicated and expensive business it generally leads to a handful companies providing the ASICs (first for themselves) leading to miner centralization. In this paper I'll present a new proof-of-work algorithm that tries to cope with this problem.

1 Introduction

In cryptocurrencies (e.g. bitcoin [3]) proof-of-work is used to guard transaction history and enforce ordering. By embedding digital signatures with a proof-of-work stamp and by sequencing proof-of-work stamps, it becomes hard and expensive to rewrite history. It is currently the only known method that is resistent to history rewrite attacks (also called "reorgnizations.") Once a comfortable amount of proof-of-work has been stacked, the users within the network can be relatively confident that their transaction can't be reversed. To ensure the network stays resistent, a difficulty parameter is also added and tuned once the network accumulates more computing power. By tuning this parameter we can ensure real-world time correlates with the proof-of-work being done.

Traditionally, the only resource being considered in proof-of-work algorithms has been computing power, but lately a new class of algorithms, such as "Cuckoo Cycle" proof-of-work [4], also enforces utilization of memory. The hope is that making the proof-of-work algorithm memory-hard makes it more difficult to create application specific circuits (ASICs) that solves the proof-of-work in much less time than conventional off-the-shelf hardware/computers. This specialized hardware typically results in making the proof-of-work network more centralized.

In this paper I'll describe a new proof-of-work memory-hard algorithm that is different from "Cuckoo Cycle" but shares some of its properties. The distinction however is by choosing a non-trivial problem that represents a whole class of existing and interesting problems, makes it meaningless to construct hardware so specialized for no other use case beyond solving the proof-of-work itself.

2 Method

The flowchart 1 provides an overview of the method being used. Given a seed for (semi) random generator we generate a star cluster (or galaxy.) This star cluster is designed so that in order to find a solution it is more or less required to keep the data points sorted in memory. In the standard design we require 2^{27} (= 134,217,728 stars) being generated which requires about 2.4 GB of memory, but the exact amount varies depending on how you choose your data structures.

The quest is to find the star constellation "The Big Dipper" N number of times (in the standard design N = 16) and each round will provide what

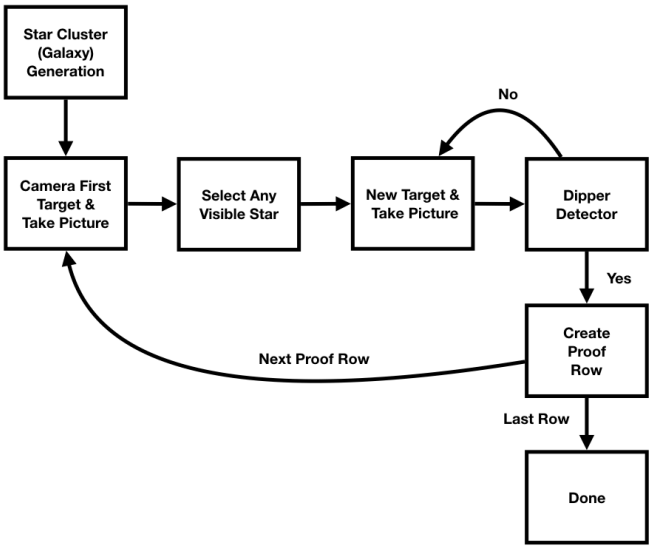

Figure 1: Algorithm Outline

is called a proof row. The final proof will consist of N rows.

The orientation of the camera is, as the generation of stars, determined by the seed value and a nonce. Everything is calibrated so the uniform distribution of stars and the sensitivity of the lens yields approximately the same number of stars being rendered regardless of how the camera is orientated and zoomed.

In each proof round, before looking for the "Big Dipper," we require the functioning proof of one visible star from the first camera target. Then the camera target is changed, and we keep changing the target until the "Big Dipper" is found. The "functioning proof" is added to prevent the user from cheating, as its details are explained in a section 8. Once the "Big Dipper" is found we can record the identifiers (32-bit) of the 7 stars that form the constellation. The entire proof round just consists of 9 32-bit integers.

Once we have a final proof it requires limited resources to verify it. We don't need to render the entire galaxy, just the stars that is given by each proof row. Thus, very limited memory (and CPU) is required to verify a proof. This is in line with other memory-hard proof-of-work algorithms.

3 Star Setup

We'll be using the $siphash(key, i)$ [2] function to compute semi-random values given a seed key and an arbitrary index value i. All parameters are 64-bit unassigned integers including its return value.

The first step is to compute a star cluster that is uniquely generated via key (or seed.) Each star is expressed in Cartesian coordinates (x,y,z) constrained into finite region of space. We'll define the position of each star as:

$$
\begin{aligned}
star_x^i &= siphash(key, 3i) && (1)\\
star_y^i &= siphash(key, 3i+1) && (2)\\
star_z^i &= siphash(key, 3i+2) && (3)\\
& && (4)
\end{aligned}
$$

The range of i ($0 \leq i < max$), (the number of stars to be generated,) is defined by the super-difficulty parameter, which I'll talk about later.

These unassigned 64-bit integer coordinates can be reinterpreted as values within $-0.5 \leq v < 0.5$ using the transformation:

$$normalize(v) = \frac{v}{2^{64} - 1} - \frac{1}{2}$$

It should be noted that in a consensus system it would be unwise in general to use floating point numbers to carry out the actual computations. Even though the IEEE 754 standard [1] defines the representation of floating point numbers, it does not define the exact behavior of all functions applied on this representation. For example, it's not uncommon to use 80-bit registers for floating point computations where the memory representation is IEEE 754 (which at most fits 64 bits.) Thus, the combination of hardware vendor and/or version of your compiler, its register allocation and spilling strategy may affect the outcome of the end result. It's in fact impossible to get 100% equal behavior on all platforms. Therefore, you should in general never, ever, use the equality operator on floating point numbers.

To overcome this floating point deficiency, all computations that are part of the consensus computation, verifies everything via either a custom fixed point representation or a custom floating point representation. As a miner you are free to use hardware floating point for your calculations, but to verify your solution you need to use the built-in fixed-point representation that is very likely much slower. But if only a solution needs to be verified,

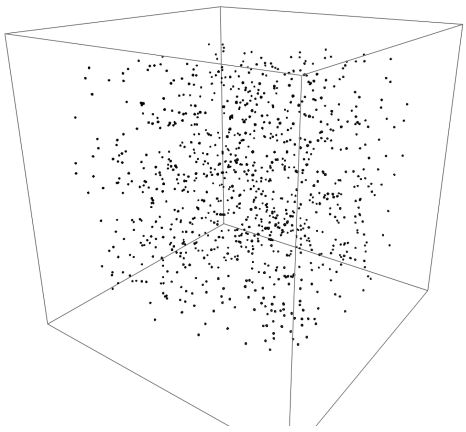

Figure 2: Stars in space

that performance degradation isn't the majority of the overall computation. Furthermore, empirically, I haven't been able to produce a solution on my hardware where the fixed-point version fails. This is because the result is verified within some tolerance. And the tolerance set is such that it should work properly for most of the time. Worst case, as a miner, you'd have to discard a detected solution and find another one.

4 Grid Partitioning

To make the problem scale invariant we'll partition the space into sub cubes, or a grid. The parameter n will determine the super-difficulty of this proof-of-work such that the number of sub cubes along each axis is 2^n. For example, if $n = 8$ we'll partition the space into $256 \times 256 \times 256$ sub cubes.

The number of stars S_n will be determined by the formula:

$$S_n = 2^{3n+3}$$

So for $n = 8$: $S_n = 2^{27} = 134{,}217{,}728$ stars (134 million.) For every $n \geq 1$ the density of stars will become 8 stars per unit or sub cube.

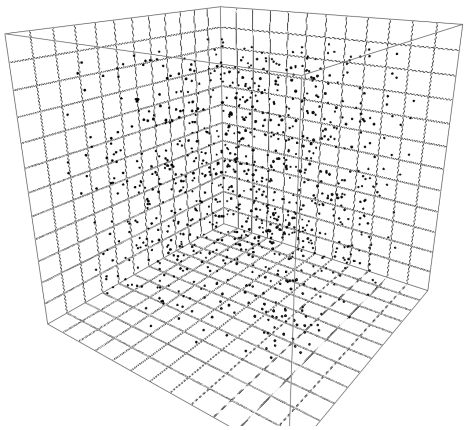

Figure 3: Space with Grid

5 The Camera

In this region of space we position a camera at coordinates (0,0,0) and pinpoint at certain coordinates (x,y,z) and project those through the lens to see if we can find "The Big Dipper" (orientation and scale invariant) within a certain tolerance.

The field of vision is captured by the volume of a pyramid (see Figure 4.) The orientation of the camera can be described by some vector algebra. If the position of the camera is $\boldsymbol{C} = (0, 0, 0)$ and the target point is $\boldsymbol{P} = (P_x, P_y, P_z)$ we can compute the direction vector as $\boldsymbol{D} = \boldsymbol{P} - \boldsymbol{C} = \boldsymbol{P}$. If we assume the existance of a proxy up vector (0,0,1) then the plane orientation of the pyramid is:

$$dir_x = \boldsymbol{D}_{norm} \times (0, 0, 1) \quad (5)$$

$$dir_y = \boldsymbol{D}_{norm} \times dir_x \quad (6)$$

Where $\boldsymbol{D}_{norm}$ is the normalized (unit) vector of $\boldsymbol{D}$, i.e. $\|\boldsymbol{D}\| = 1$, or

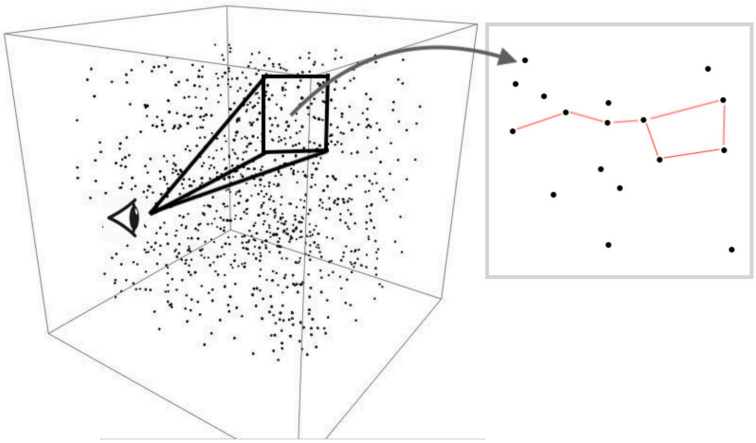

Figure 4: Looking for Big Dipper

$$D_x = \frac{D_x}{\sqrt{{D_x}^2 + {D_y}^2 + {D_z}^2}} \tag{7}$$

$$D_y = \frac{D_y}{\sqrt{{D_x}^2 + {D_y}^2 + {D_z}^2}} \tag{8}$$

$$D_z = \frac{D_z}{\sqrt{{D_x}^2 + {D_y}^2 + {D_z}^2}} \tag{9}$$

If we fix the volume of the pyramid, then the vision field/volume will be the same regardless where the target point is located. Therefore, if the camera zooms in on an object, the lens will become more sensitive and the end result is that the same number of stars will approximately be projected regardless of chosen target.

Let's make the volume constant to be $V = 32$ (sub cubes) then we'll approximately see $S_V = 8 \cdot 32 = 256$ stars. The height h of the pyramid is $\|\boldsymbol{D}\|$, recall that $\boldsymbol{D} = \boldsymbol{P}$ and the using the formula of the volume of a pyramid we can compute the volume base length as: $l = \sqrt{\frac{32*3}{h}}$.

By sorting the stars into sub cubes, we can quickly approximate the sub cubes that need to be considered, basically moving along the direction vector and consider the points within the pyramid. For each star within each relevant sub cube we calculate:

$$S_x = (\boldsymbol{S} \cdot dir_x)\frac{\|\boldsymbol{P}\|}{l\|\boldsymbol{S}\|} \tag{10}$$

$$S_y = (\boldsymbol{S} \cdot dir_y)\frac{\|\boldsymbol{P}\|}{l\|\boldsymbol{S}\|} \tag{11}$$

S_x and S_y is the the projection of the star, and it is on the display if $-0.5 \leq S_x < 0.5$ and $-0.5 \leq S_y < 0.5$ (same as ${S_x}^2 < 0.25$ and ${S_y}^2 < 0.25$.) We also need to check that $\|\boldsymbol{S}\| < \|\boldsymbol{D}\|$.

6 The Detector

Once we projected all the stars on the screen the next step is to make the detector that finds "The Big Dipper." In the previous section we've computed the projection of all stars on the screen, let's denote those as the set Σ.

The algorithm is relatively naïve. For every pair of stars $(s_i, s_j) \in \Sigma$ we assume it matches the first two stars in the "Big Dipper." From this we can compute the rotation and scaling matrix that reorients the "Big Dipper" into this position. Then we look for stars within the remaining spots (by using a template of the "Big Dipper" and apply the rotation & scaling matrix.) If we can find all the stars of "Big Dipper" within a specified tolerance, then we're done. Let's define delta of the two stars as $ds_{ij} = s_j - s_i$ and the delta of the two template stars $dt^{12} = t_2 - t_1$, then rotation and scaling matrix becomes:

$$R = \begin{bmatrix} \frac{\|ds^{ij}\|(dt_x^{12}ds_x^{ij} + dt_y^{12}ds_y^{ij})}{\|dt^{12}\|} & \frac{\|ds^{ij}\|(dt_x^{12}ds_y^{ij} - dt_x^{12}ds_y^{ij})}{\|dt^{12}\|} \\ \frac{\|ds^{ij}\|(dt_x^{12}ds_y^{ij} - dt_x^{12}ds_y^{ij})}{\|dt^{12}\|} & \frac{\|ds^{ij}\|(dt_x^{12}ds_x^{ij} + dt_y^{12}ds_y^{ij})}{\|dt^{12}\|} \end{bmatrix} \tag{12}$$

The remaining position of the other stars can be computed as:

$$\begin{pmatrix} x \\ y \end{pmatrix} = R\begin{pmatrix} dt_x^{1k} \\ dt_y^{1k} \end{pmatrix} + \begin{pmatrix} s_x^i \\ s_y^i \end{pmatrix} \tag{13}$$

We've chosen a really tight tolerance as it is quite easy to find the "Big Dipper" otherwise. If the screen is projected onto 4096x4096 pixels, then each star position needs to more or less be within one pixel's distance. Of course, the tolerance is scaled as well with $\frac{\|ds^{ij}\|}{\|dt^{12}\|}$.

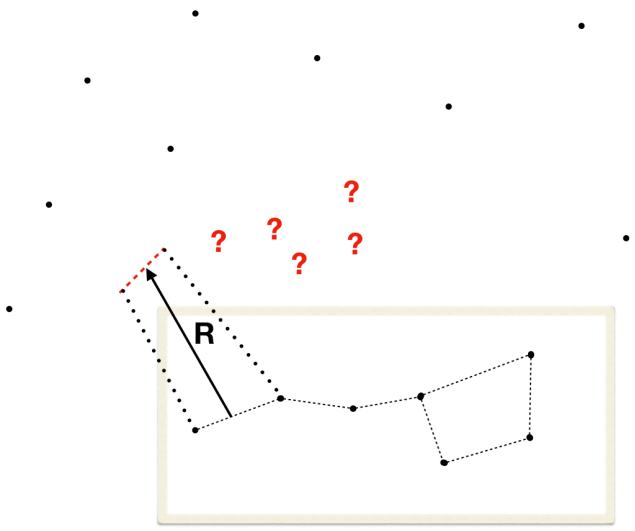

Figure 5: Detecting Big Dipper

7 Scanning

The camera is fixed at origin (0,0,0) and the target point is defined by:

$$P_x(M, I) = siphash(Key, M2^{32} + 3I) \quad (14)$$
$$P_y(M, I) = siphash(Key, M2^{32} + 3I + 1) \quad (15)$$
$$P_z(M, I) = siphash(Key, M2^{32} + 3I + 2) \quad (16)$$

Here I is the iteration (nonce) and we attempt to find the smallest I such that the "Big Dipper" is found using the detector algorithm described in the previous section.

M is the nonce offset and the final nonce is computed by adding it with a factor of 2^{32} to I. The nonce offset is computed so it is the sum of all previous nonces of the previous proof rounds (0 if we run the first proof round.) The idea here is to make it as difficult as possible for the user to guess what region of space we're browsing. Otherwise, the miner could precalculate the region of space of interest and optimize the data structures to reduce memory consumption. As this proof-of-work is supposed to be memory-hard it would defeat its purpose otherwise.

8 Verification

For each proof round, it can be quickly verified by 9 (32-bit) integers. 7 for the identification numbers for the stars that form the constelltation ("Big Dipper".) The first two integers are:

1. One 32-bit integer to represent the identification number of a visible star. (Camera nonce is computed from the sum of all previous nonces, or 0 if it's the first proof round.)
2. One 32-bit integer to represent the nonce for which the "Big Dipper" is found.

As only 8 stars need to be rendered it is farily quick in verifying that the dipper has been found using limited resources. We use 16 proof rounds for a final solution (i.e. $0 \leq N < 16$.)

In table 8 we've shown the complete proof for seed "hello42." Note the nonces in the second column are nonces found within that proof row. The nonce used for the camera is: ((row number + Σ previous nonces) $\times 2^{32}$ + nonce) for the 7 stars. And the nonce used for the proof of first visible star is: (row number + Σ previous nonces) $\times 2^{32}$.

Table 1: Final proof table for seed "hello42"

Star Id	Nonce	Star Id 1	2	3	4	5	6	7
45134394	73	60685796	33167674	121403207	105008779	26799832	68979659	41313803
121366534	445	27556923	39213525	69648547	60924222	24065809	67739412	5626835
61387993	5861	94168951	71280049	107176517	38931956	45346251	77699840	92351466
64700296	1464	95125960	113171749	30245602	76154893	4523885	25269435	73419704
19920804	1636	52287909	50693065	43134645	51857309	6273341	5692871	67184462
14222398	252	98719556	88585662	125624358	8666191	103225255	133958251	22853665
129242039	3536	119831471	53378726	23013397	114073381	7072602	51229030	93605232
29242642	458	53013453	69243949	46610740	131863297	43795259	78472623	72407954
10761019	1423	88958339	31170727	101940907	18896769	20899400	32463797	74429216
44472422	2254	87095471	57447590	43033310	129969678	101793249	105999249	1135863
104734877	1818	101095787	95874187	118610637	127618294	88003971	121905569	35849384
34912470	1566	19331311	24911744	64392351	100928396	21458762	52290402	58839198
91025279	111	88433232	113922369	67329741	83592379	79006603	71019432	118722747
18454763	1386	123182890	43387352	37814584	112260055	9076680	118476458	34795363
94824887	3760	88553642	37167773	38666071	126411100	60878172	86885815	65116994
100215655	63	21844082	31560854	57639403	115560688	121125789	126761850	31520447

9 Fine Tuning

As computers become faster, or more computers are joining the proof-of-work network, we need a parameter to fine tune its difficulty. The easiest way to achieve this is to put an additional constraint on the final solution. Let's say: $SHA256(solution) < H < 2^{256}$ where a descreasing H will make it harder to find an accepted solution. In essence, this is the strategy used in bitcoin [3] but here used as an additional constraint on the "Big Dipper" solution itself. (Same approach is taken in "Cuckoo Cycle.")

10 Course Tuning

We can set the super-difficulty using the parameter n. The setup is tuned so that only more memory is consumed while having all the other properties constant.

Using $n = 8$ consumes approximately 2.4 GB of memory, but is of course dependent on how you build up your data structures and what kind of optimizations you apply. $n = 9$ consumes 8 times more memory, so around 20 GB. The problematic aspect of just increasing n is that specialized hardware may make it difficult for off-the-shelf hardware to compete. Thus, although the current framework makes it possible to tune super-difficulty (automatically as well), it's determined that this is may be an unwise idea.

If specialized hardware comes online that solves the "Big Dipper" proof-of-work algorithm fast, it's likely that the same hardware can be used for general purpose vector algebra computations on big datasets. To quickly render 2.4 GB of memory using floating point operations with vector algebra is likely to be useful for many other applications such as computer graphics and/or computational physics. Therefore, if the hardware has other

use cases, it'll increase the chances of making that hardware more general purpose.

11 Conclusion

This proof-of-work algorithm is far from trivial. It is designed to consume a lot of memory and computation using floating point numbers. It is hard to come up with specialized hardware for computing this specific proof-of-work only, rather it makes more sense that incorporates computations on big datasets with floating point numbers and vector algebra in general; something that is useful for a big class of problems, e.g. computer graphics and/or computational physics. Thus, if specialized hardware is pursuited it is more likely to become off-the-shelf hardware for other applications as well. This by itself should reduce, but not eliminate, miner centralization.

References

[1] *IEEE standard for binary floating-point arithmetic*. Institute of Electrical and Electronics Engineers, New York, 1985. Note: Standard 754–1985.

[2] Jean-Philippe Aumasson and Daniel J. Bernstein. Siphash: A fast short-input PRF. In *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, pages 489–508, 2012.

[3] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, May 2009.

[4] John Tromp. Cuckoo cycle: a graph-theoretic proof-of-work system, 2014.