Game Theory and Probability with Bitcoin (First partial draft: Chapters 1 - 7. Please send questions or comments to micahw@uoregon.edu)

Micah Warren

September 3, 2021

Contents

Introduction

The question of how one person can transfer money to someone else without physically exchanging money is an old one. The most common solution, available for hundreds of years, is to use a bank. If a bank is mutually trusted, and has custody of one person's money, this person can write a check, or by some other means, direct the bank to pay another party. A check can be thought of as simply instructions for the money's custodian to do something with the money.

Now this requires trust. If the two parties do not use the same bank, there must be a relationship between the banks. If the banks are in different jurisdictions, this can be an obstacle.

For many reasons, individuals may want to transmit money without the use of a trusted third party. They may want to do this at a distance. They may also want to do this with some level of anonymity. Suppose a network of individuals forms around this goal: To be able send money to each other, without doing so in person. Keeping in mind the notion of a bank cashing a check, there are two issues that must be overcome:

- There must be a way for everyone involved in the network know what money is good, at any given time.
- Parties must be able to "sign" payments in a verifiable way. That is, there must be some way to verify that the instructions to transfer money are coming from the owner of the money.
- There must be a way to insure that a payment is not reused repeatedly.

Crytpography (number theory) solves the second issue quite handily. A blockchain attempts to solve the other two problems, with probablistic assurances.

The goal is this text is to offer some game theoretic background for analysis of how Proof of Work blockchains.

The backgroud material provided in the first three chapters will begin with a moderately detailed discussion of how Proof of Work functions, then an overview of basic notions, of Game Theory, followed by some relevant notions of Probability.

In the remainder we will discuss the implication for Bitcoin.

This textbook began as a syllabus for the Clark Honors College at University or Oregon, proposed in February of 2020. The course has yet to run (it was intended to be a discussion format class, probably not best for remote teaching) but as I developed much more material than would fit in a one quarter discussion course, I realized this would fill a textbook that could serve as an upper division course in Mathematics, Computer Science, or Economics. My hope is for this text to have three application:

- To be a reasonable textbook for such an upper division university course.
- Estabilishing a foundational reference framework for experts in the blockchain industry discussing game theory.
- Provide a relatively thorough reference for journalists, policymakers or other who may have little exposure to blockchain technology but would like to understand the underpinnings of blockchain security.

I have tried to make it largely self-contained, but it's impossible to make it 100% so, especially because the material draws from a broad set of topics. Some terms maybe unfamiliar, but if the Wikipedia entry is clear enough i have not made an effort to include extra. For example if you need to know they full definition of "rooted tree" you can easily find this information, and the context in which you find it will not be sufficiently different. Therefore I don't find it worth it to tailor a definition for this text. Wikipedia is your friend, as is Bitcoinwiki and many other sources. (Although as always, be careful, a blogger who attempts to define a term is not necessarily an expert, regardless of their popularity.) Some linear algebra is assumed, but if you are unfamiliar, I have provided words that should make tracking the definitions down easy for anyone with access to a search engine.

Chapter 1

Basic of cryptographic payments

We begin with an extremely short overview of cryptographic signature

1.1 Preliminaries on cryptographic signatures

1.1.1 Binary and Hexadecimal

The standard number system we know and are used to is base 10. This requires 10 digits and allows us to represent numbers using these digits, for example

$$328 = 3 * 100 + 2 * 10 + 8 * 1.$$

Each digit corresponds to a higher power of ten. For example, written as powers of 10, we have

$$43729 = 4 * 10^4 + 3 * 10^3 + 7 * 10^2 + 2 * 10^1 + 9 * 10^0.$$

(Recall that $10^0 = 1$.)

Now there is nothing special about 10, other than perhaps it corresponds to the number of fingers (which are also known as digits) that most people have. A computer does not have 10 fingers, instead computers build data with two digits, 0 and 1. So computers prefer to store numbers in binary. Binary numbers will be strings that consist of only 0 and 1.

An example of a binary number

$$1001 = 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0$$

which is more familiar to base 10 users as the number 9. To count in binary, starting with 0, we have

$$0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, ...$$

A 4-digit binary number represents any number between 0 and 15 (so $2^4 = 16$ total numbers). An 8 digit binary number will correspond to numbers betwen 0 and 255, giving 2^8 possibilities.

Now since computers use binary, when dealing with integers, it makes architectural sense to deal with sets of integers that are powers of 2. We can compress this further. Noting that $2^4 = 16$ we can convert each 4-digit binary number into a number between 0 and 15. In order to do this, we take the standard numbers, 0–9,and then take A,B,C,D,E,F to represent the remaining 6. This gives us 16 digits, so we can write numbers in base 16, also know as hexadecimal.

Binary	Hexadecimal	Base 10
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Hexadecimal is especially convenient when writing huge numbers.

Hex	Base 10
$FFFF$	65636
1000000	16 777 216
$FFFFFFFF$	4294 967 295
100000000	4294 967 296
$FFFFFFFFFFFFFFFF$	18 446 744 073 709 551 615

.

Note that a 256 digit binary number, which will represent the numbers between 0 and

$$2^{256} - 1 = 115\,792\,089\,237\,316\,195\,423\,570\,985\,008\,687\,907\,853$$
$$269\,984\,665\,640\,564\,039\,457\,584\,007\,913\,129\,639\,936$$

can also be represented by a 64 digit hexadecimal (because $2^{256} = 16^{64} \approx 10^{77}$). Many of the frequently used algebraic function involved in cryptography will involve 64 digit hexadecimal numbers.

1.1.2 Public-Key cryptography

As it's not within the scope of this course, we avoid the temptations to do a deep dive into number theory. However we will discuss public-key protocols in general terms.

The idea of a public-key (or asymmetric) encryption scheme is the following. It's possible to generate two passwords p and q that have a special relationship: If you encrypt a message with p it can be decrypted with q. While the pair (p, q) is easy to generate, it would be very difficult to determine q, simply by knowing p. What this allows one to do is publish the password p. This will be called the "public key." If you keep q a secret, anybody will be able to send you a message encrypted with p and only you will be able to decrypt it, because only you know the decryption password q.

For example, suppose Alice generates the pair

$$p = F03D12A2$$
$$q = 1ED77138.$$

Alice then publish the information "$p = F03D12A2$." Bob would like to send Alice a secret message "Happy Birthday." He makes use of an encryption function E, which takes a message M, and an encryption password p, and spits out a garble of nonsense, $E(M, p)$. In this case Bob computes

$$E(\text{"Happy Birthday"},\ \text{F03D12A2}) = \text{AF9817314741109B}.$$

The string "A7314741109B" should be nonsense to most people. However, because Alice has the appropriate decryption password q, she can use the decrypting function D and compute

$$D\,(\text{AF9817314741109B,1ED77138}) = \text{"Happy Birthday"}.$$

If someone were to try to decrypt the message with the wrong q, they would get something of the form

$$D\,(\text{AF9817314741109B,6F899D68}) = \text{"G*hAaUytbaQ1"}.$$

Again, this works because it's relatively easy to generate p and q at the same time, but essentially impossible to guess q if you know p.

This allows for two sorts of operations:

First, Alice may generate a pair (p, q). If she publishes the password p and keeps q a secret, then anybody will be able to use $E(M, p)$ to encrypted a message to Alice, but only Alice will be able recover

$$M = D(E(M, p), q)$$

This is described above.

On the other hand Bob may generate a pair (p, q) and publish q and keep p a secret. Thus, if Bob encrypts a message $E(M, p)$, anybody is able to decrypt the message using q, which is public information. However, only Bob will be able to encrypt messages that can subsequently be decrypted using q. In this manner, Bob can "sign" a message. You can verify Bob's signature by decrypting using q. If you decrypt a message using q and you get nonsense, the message was not signed by Bob.

Next, note that these two can be combined. In particular, Bob, can encrypt a message M using Alice's public encryption password, and then encrypt that message with his own private password. When Alice recieves this message, she can first decrypt it using Bob's pubic decryption password (which will result in nonsense) and then decrypt it again, using her own private decryption password. The result is that Bob can publicly broadcast a message that only Alice can decrypt, and Alice can also verify that it is from Bob and not someone impersonating Bob.

Typically, these keys p and q are very long. A Bitcoin address is a 160-bit hash of the public portion of a public/private keypair. Recall that there are $2^{160} \approx 10^{48}$ possible 160 digit binary numbers. This is more than Avogadro's number squared. In short, even with the world's most powerful computers it should take millennia to test enough p to find the right one.

Hashing function.

A hashing function take a message, string, or number, and converts this into a number in such a way that it would be nearly impossible to invert. For example, the RSA hashing algorithm (which used as a well-known public key protocol) works as follows. Take a pair of large numbers (n, e). Then given a "message" m, which will be a number between 0 and n, compute the following

$$E(m) = m^e \bmod n \tag{1.1.1}$$

where the notation

$$a \bmod b$$

(read "a modulo b") means "take the remainder when dividing a by b." For example,

$$19 \bmod 4 = 3$$
$$27 \bmod 10 = 7.$$

In the example (1.1.1), it is easy to see that for large numbers e and n, the hashed value will be nearly impossible to predict. In other words, while it is easy to directly compute (for a computer built after 1975) in the forward direction, say

$$4219^{1687} \bmod 67108 = 43\,511$$

it would be very difficult to go backwards, and determine who m is by solving the equation

$$m^{1687} \bmod 67108 = 43\,511.$$

It's easy to verify that 4219 is a solution, but it is not easy to find 4219. Because these numbers are relatively small, it would take most computers less than a few seconds to find 4219 by brute force. However, if we were to take n and e to be numbers on the order of 10^{77} (Note that 10^{82} is an estimate for the number of atoms in our observable universe) it would be essentially impossible to find m by brute force.

Takeaway

It is possible for anyone (using basic software) to generate a public and private keypair. Using the public key, anybody can verify that a message was encrypted with the private key, without knowing the private key. This allows the opportunity for Bob, whose public key is known, to sign a message, saying that "I would like send Alice 1 Bitcoin." Everybody can read the message and verify it is from Bob.

In the most basic sense, this is how Bitcoin works. If Bob has a Bitcoin, he can sign a message declaring that he is transfering that Bitcoin to Alice. Now everybody can verify that the Bitcoin that once belonged to Bob now belongs to Alice. Subsequently, if Alice would like to send that Bitcoin elsewhere, she simply signs a message, saying, "The Bitcoin I got from Bob, I now send to Charlie."

Technically, Bitcoin uses what is called a UTXO protocol, which means it is not account based (like, say a bank) but rather transaction based. This means that instead of having an account number, you have various private keys each that control the funds in certain transaction. Each transaction must come from a previous transaction. This means that each fraction of a Bitcoin can be traced backwards all the way to where it was created.

1.2 Blockchain Protocols

The ability to use cryptography to sign a messages has been available for decades. Naturally, the question follows of how to use this method for transferring ownership. By itself, this not enough to create a system of money. The most obvious problem is what is known as the "double-spend" problem. Namely, suppose Bob sends a digital coin to Alice, and then later that day Bob sends the same digital coin to Charlie. Both transactions appear to be valid on their own. Wherever Bob received his digital coin, he is assigning it now to Alice, and he is also assigning that same coin to Charlie. This would cause a problem if Bob was using this digital coin to buy something. For example it would appear that Bob paid Alice. But Bob could then pay Charlie with the same digital coin. Hence the name "double-spend."

In order to solve this problem, there has to be a way to determine which transaction is valid, in a such a way that everyone in the network agrees.

1.2.1 Consensus

Suppose a group of friends would like to meet up at a restaurant. In order to do this, they must communicate with each other to ensure that they all know at which restaurant they are going to meet. Consensus is not simply the act of negotiating. While some of the friends may have different food preferences, consensus is achieved when, and only when, all of the friends know which restaurant they are headed to. Consensus doesn't make any normative judgments.

To illustrate the difference between consensus and negotiation, suppose friends (without cell phones) plan on meeting up tomorrow for a drink. They make the following decision. If it's sunny and nice, we will meet at the patio at Magoo's. If it's cold and rainy, we will meet at the lounge of The Moon Temple. As the time approaches, a few clouds form, and a few small raindrops fall from the sky. While the patio at Magoo's may still be a suitable place, what the friends face now is a consensus problem, not a negotiation problem. They need to determine a way in which all parties can decisively determine if it's raining. They must decide "rain" or "sun." This can be difficult for a number of reasons. For example, if a single raindrop is to be interpreted as "rain", then it's always possible that one person will detect rain while another will not. Thus it isn't always clear that everybody will feel rain if the rain is sparse.

Consensus protocols are everywhere. When the quarterback for a football team steps up to the line of scrimmage, the hand signals and code calling is for the immediate concern that the football team all agrees on the same play, not to worry about the optimal play. If the team all agrees on what play is going to be run, they have a chance. If they don't, it will likely be a disaster. Similarly, sports teams use referees or umpires as a consensus mechanism. Clearly a baseball game could not go forward if the two teams did not agree as to whether a player was safe or out.

Similarly, the modern social contract of law and order uses a consensus protocol. While we may not all agree on whether the law is just, moral, equitable or optimal, we can at least hope to agree on 1)what the law is and 2) what procedure is required to change the law. The social contract is based on the idea that while we may not agree on what the law should be, at least we agree on what the law is.

A classic question of consensus is the Byzantine General's Problem: Suppose that several divisions of an army are camped around a city. They must come to an agreement on how to launch an attack. As for the football team, their success will be much improved if they agree on an attack. An coordinated attack is bound to fail.

For applications on a network, nodes often must agree on a binary proposition in order to proceed. In the double-spend example, the network doesn't really care whether Bob has sent a specific coin to Alice, or that same coin to Charlie. What the network wants is clarity. Either the transaction to Alice is valid or the transaction to Alice is invalid. As long as the decision has been made, the network can continue its business. Alice can then spend the coin, and Charlie knows he doesn't have the coin.

Generally consensus protocols do are not concerned with the agreement being just, factually correct, optimal, moral or anything else. The fundamental goal of consensus is to come to an agreement, not to necessarily make a judgment on which transaction "should" be the right one. The object is to determine, in a relatively short time whether a given transaction is valid or not. Once a transaction is determined to be valid, it is desirable for everyone to expect that it will remain valid forever.

The simplest means to consensus is often to choose a leader. For example, on a football team, the quarterback calls the plays, and nobody argues. This is efficient, but is obviously antithetical to the nature of a decentralized payment network like Bitcoin. If there was a leader, the network could be shut down by eliminating the leader, introducing a single point of failure. A leader could also choose to abuse their power, for example by accepting bribes from Bob or Charlie. A decentrailzed network does not want to rely on the continued benevolence of one entity.

Fun Problems

Problem 1. *Consider a network of consisting of an unknown number of nodes, say $3<N<1000$. At some point in time, each node is handed a random bit 0 or 1. The network then has to all decide collectively on 0 or 1. Devise some protocols the nodes could agree upon ahead of the time for choosing 0 or 1, subject to the following criteria*

- All nodes are treated the same way at the start. That is, we can't pre-assign a leader.
- Nodes can broadcast information to the group
- It must be possible for the agreement to be either 0 or 1. (That is, we can't simply decide ahead of time the answer is 0).

Problem 2. *Prisoner's light bulb problem: There are 100 prisoners in solitary cells. There's a central living room with one light bulb; this bulb is initially off. No prisoner can see the light bulb from his or her own cell. Everyday, the warden picks a prisoner equally at random, and that prisoner visits the living room. While there, the prisoner can toggle the bulb if he or she wishes. Also, the prisoner has the option of asserting that all 100 prisoners have been to the living room by now. If this assertion is false, all 100 prisoners are shot. However, if it is indeed true, all prisoners are set free and inducted into MENSA, since the world could always use more smart people. Thus, the assertion should only be made if the prisoner is 100% certain of its validity. The prisoners are allowed to get together one night in the courtyard, to discuss a plan. What plan should they agree on, so that eventually, someone will make a correct assertion?*

- Good luck. You can google the solution.

1.2.2 Proof of Work

In 2008, in the paper "Bitcoin: A Peer-to-Peer Electronic Cash System," Satoshi Nakamoto proposed a solution to the double-spend problem using a blockchain and Proof of Work.

A blockchain, as the name suggests, is simply a sequence of blocks. The blocks necessarily go in an order. The genesis block, which contains an initial accounting of which coins are controlled by which private keys, is usually lablled as block 0. The next block will be block 1, and so forth.

Each block is simply a list of transactions. Necessarily, the transactions must be valid to be included in the block. In order to be valid, the transactions in block $N+1$ cannot contradict any transactions contained in blocks 0 through N. This ordering will solve the double-spend problem. If Bob sends a bitcoin to Alice in block 2178, he cannot then send the same bitcoin to Charlie in block 2192 - such a transaction will be considered invalid because it contradicts a previous block.

So essentially, the Bitcoin blockchain is simply a very long database of every Bitcoin transaction that is considered valid. Necessarily, these transactions are not contradictory, if they fit on the same blockchain.

At first, this may sound like a straightforward mechanism. Simply order the transactions and then all will be clear. However, in practice this is not simple. There needs to be a very decisive way to determine the next block so that all parties in the network can agree on what the next block is. If each node could submit their own block, this could spell potential trouble, especially as nodes throughout the world attempt to communicate. There would be no consensus, and the network could not move forward. So the Bitcoin whitepaper [Nakamoto2008] uses what is called Proof of Work, based on the following principles.

- Whoever creates the block earns a reward (in bitcoin) for creating the block. This incentivizes parties to create the block
- It is somehow expensive or difficult to create each block
- Blocks are so difficult to produce that they only appear about every 10 minutes.

The block producers are called miners. To produce a block, a miner does the following.

1. Compile a list of valid transactions compatible with the current blockchain.
2. Generates a large random number, called a nonce ("number used only once").

3. The nonce is combined with the list valid transactions and hashed using a hashing function. The hashing function takes the input and generates an integer, often written as a hexidecimal number in a very large range, which looks something like
$$b0d01118e5480f5df5c97eea94405c80bac5cd3c8fa638541aa6dcfa773cc76b. \tag{1.2.1}$$
4. The miner checks to see if the number is below a certain known threshold called the **target**. Usually this means is has quite a few leading zeros, and looks something like
$$0000000000000000000 2dac9061b57715ff6821cd03e13e49d026912fea3fe70 \tag{1.2.2}$$
If the number is small enough, below the target, then the miner rejoices, having created a new block. They then broadcast the block, together with the nonce and hash, to the network. This becomes the next block on the blockchain. The miner is given a reward of a specified number of bitcoins, which as of 2021, is 6.25 bitcoins.
5. If the number created by the hash function was not below the threshold. Go back to step 2 and repeat.
6. Repeat trillions of times until you find a hash that meets the threshold, or, another miner broadcasts that they have found a block. In this case, add the block to your blockchain and go back to step 1.

Look at the two hexidecimal numbers above. The number (1.2.1) is more or less a random string of digits, while the latter one (1.2.2) leads with quite a few zeroes. (This was an actual hash of a block in May 2021). Because the hash function is generating numbers that are more or less random between 0 and $2^{256} = 16^{64}$. The chance of getting 19 leading zero is about 1 out of $16^{19} \approx 75 * 10^{21}$. This means that if you perform $75 * 10^{21}$ hashes, the expected number of hashes with 19 leading zeros is 1.

As of May 2021, the "hashrate", that is, the totally number of hashes that all miners are producing is about 180 million TH/s (terahashes per second). Recalling that a terahash is a trillion hashes, this means that about $180,000,000,000,000,000,000$ hashes are being performed every second. There's 600 seconds in 10 minutes, so we conclude that

$108\,000 \times 10^{18} = 108 \times 10^{21}$ are performed every ten minutes. Notice that 108×10^{21} is rather close to the number $75 * 10^{21}$, which is how many hashes it takes so that the expected number of hashes whose first 19 digits are zero is at least one.

Miners are often organized into **mining pools**. Because a block is only rewarded about once every ten minutes, if there were 100,000 individual miners mining, the average miner would win a block about once every two years. It makes more sense then to pool resources and split the profit, so in practice there will be a dozens of mining pools that control a large enough fraction of the computing power so that they can expect rewards on a regular basis.

Each pool has it's own hashrate, and the fraction of the total hashrate commanded by that pool determines their probability of winning the next block. For example if Pool A is producing 12 million TH/s, and the total hashrate is 180 million TH/s this pool has a probability of

$$p = \frac{12 \times 10^6 \text{TH/s}}{180 \times 10^6 \text{TH/s}} = \frac{1}{15}$$

of winning any given block, or about a block every two and half hours.

The above quotient is as simple as it looks: The probability of winning the next block is always going to be given by the fraction of hashrate the pool controls. This simple expression doesn't unveil the mathematics of independent Poisson processes that lies underneath.

A **Poisson process** is a randomly occurring event such that the average frequency of the event over time can be predicted, however is not expected to happen periodically or with any regularity. Each occurrence of the event is independent of previous occurrences. A natural example is earthquakes. A 2020 study *(Bulletin of the Seismological Society of America) suggests there is 2.3% chance of a 7.5 earthquake to occur in California in the coming year. On average this is slightly more than 1 every fifty years. This does not mean, however, that if we haven't had an earthquake in the last 40 years the probability has gone up. The probability of an event governed by a Poisson process happening in the next year is the same as last year, whether or not the event happened last year. Similarly, someone who plays lottery every day has the same chance of winning every day, and does not "build up" an expectation of winning.

Mining works the same way. Each second a miner is producing trillions of hashes, each of which has a low probability of being below the difficulty threshold. Each hash output is independent of the hash that came before it, or after it, or any of the hashes produced by competing miners. For a mining pool that controls 1% of the hashrate, they would expect on average to produce a winning hash each 1000 minutes.

In our analysis it is often important to consider the probability of getting the next block. The second that a new block is broadcast, all miners (at least under typical operation) will immediately find transactions that have yet to be included and attempt to build the next block. **The probability that a given pool wins the next block is precisely the fraction of the hashrate that pool commands.** For mining games, as we will see, this fact is extremely important because winning the next block or blocks after that can be a game-changer, literally.

This is not a statement about how long one would expect the next block to take.

The interesting takeaway to note is that while the number of block produced in a time period may vary, the expected allocation to each miner is alway the same. So even if one mining pool is quite lucky and produced multiple blocks within a few minutes, the other miners chances of getting a block are not diminished.

So in a fixed time period the miners competition is not so much against each other, but against randomness. However, over a large period of time, the competition is with the average miner in recent history. This is due to the difficulty adjustment.

Block Difficulty adjustment

Computers become faster every year, and Bitcoins have on average become more valuable over time. So the Bitcoin protocol readjusts the difficulty every 2016 blocks. The difficulty measures your odds of computing a hash below the threshold. The target is a number inversely proportional to the difficulty. If the target decreases, say decrease by a factor one half, this corresponds to the difficulty increasing by a factor of two. Every 2016 blocks, the average time to compute these blocks is computed. If this number is less than 10 minutes, then the difficulty is multiplied by a factor larger than 1, with the idea that in the upcoming blocks, the average time will stay close to 10 minutes. Similary, if the average time is more than 10, the difficulty is lowered (or equivalently, the target threshold is raised.)

Longest chain rule.

Occasionally, two blocks may be produced independently, both appending different blocks to the same block. In this case, there is what's called a "fork". Miners then have an option, build on one chain, or build on the other. The consensus protocol in Bitcoin is that "the longest chain is the valid chain." This was how it was described in the original whitepaper by Satoshi. This means that as soon as one of the miners produces a block that extends one of the competing blocks, that chain becomes "the" blockchain. At this point, most miners will ignore the other chains and all mine on "the" chain.

There is technical issue here: "Longest chain" cannot literally refer to the chain with largest number of blocks. If this were the case, it's feasible for a fork to nearly catch up while using much less work by abusing the difficulty adjustment. If there was to be a fork and one chain was mined by only 1% of miners, while the other chain was mined by 99% of miners, even after many years the number of blocks would be roughly equivalent due to difficulty adjustment. So what's necessary is a more precise quantity called "chainwork". Chainwork measures the of the total a difficulties overcome in order to create the particular chain. While the whitepaper used the term "longest chain" it would be easy to catch up by using much less work, mainly because the difficulty adjustment is meant to slow down blocks produced when the hashrate speeds up.

The chainwork accumulated with each block does not depend on the particular hash of the block, only the difficulty that was surmounted. This means that two blocks that are produced at the same time will have the same accumulated chainwork, regardless of which block enjoyed a lower hash. If this were not the case, when a block is produced with hash falling just below the target would be vulnerable to being scooped by a block which equalized the chain length but had a lower hash rate. This works against the consensus. Instead, the convention is to use the "first-seen." Miners, by convention will take the first block the see, and begin extending that. There is no "rule" requiring this, but this is default.

Using the "first-seen" convention, forks that naturally occur when two block are produced at about the same time will usually be sorted quickly.

Halving.

Approximately every 4 years, the reward for producing a block is cut in half. In 2009, the block reward was 50 Bitcoins. This was cut in half in November 2012 to 25, then 12.5 Bitcoin in July 2016 , and most recently to 6.25 in May 2020 . Note that 4 years is measured in blocks, namely 210,000 blocks. Because the blocks often are produced at a slightly higher rate than 1 every ten minutes, the halvings can run slightly ahead of schedule.

Note that the block reward is the only way that new bitcoins are produced. The number of bitcoins produced in between July 2016 and May 2020 was $12.5 \times 210000 \approx 2.6$ Million. The number produced between May 2020 and the next halving will be $6.25 \times 210000 \approx 1.3$ Million. This number will continue to be cut in half every 4 years. This means that the number of Bitcoins mined forms a geometric series:

$$50 \times 21000 + \frac{1}{2} \times 50 \times 21000 + \frac{1}{2} \times \frac{1}{2} \times 50 \times 21000...$$

or in sigma notation

$$\begin{aligned} &\sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k 50 \times 21000 \\ &= 50 \times 21000 \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k . \end{aligned}$$

You may recall the formula for geometric series:

$$\sum_{k=0}^{m} x^k = 1 + x + x^2 + ... + x^m = \frac{1 - x^{m+1}}{1 - x}$$

which can be verified by doing polynomial long division on the expression $\frac{1-x^{m+1}}{1-x}$. Now because

$$\left(\frac{1}{2}\right)^{m+1} \to 0 \text{ as } m \to \infty$$

we can take a limit:

$$50 \times 21000 \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k = 50 \times 21000 \times \frac{1}{1-\frac{1}{2}} = 21 \times 10^6.$$

Thus the total supply of bitcoins every produced is 21 million. Practically, because the block reward halves every 4 years, roughly speaking, after 30 more halvings, the block reward will be

$$6.25 \times \left(\frac{1}{2}\right)^{30} \approx 5.820\,8 \times 10^{-9}$$

bitcoins. This should happen a little before the year 2140. Because the smallest denomination of bitcoin (called a satoshi) is 10^{-8} bitcoins, it will be impossible, by current accounting, to continue to pay miners by blockrewards, and the block reward will cease. However it is expected that transaction fees will fund miners. (This will be discussed later in sections ??)

Why is it called "Proof of Work"?

The goal was to create a significant barrier to writing the block, so that only miners who put in enough effort or spent enough electricity would receive the reward. Now certainly the amount of effort put forth is not easy to directly verify, especially considering the network should be open to potentially anonymous parties throughout the world. It doesn't make sense for the miners to "prove" they've computed trillions of mathematical hashes by spitting out all of the hashes, as this would take just as much effort to verify. So instead, Proof of Work asks miners to repeat a computation trillions of times, and only report to the network when they find the needle in the haystack. If the odds are astronomically small each time you repeat the calculation, then probability theory says that you must repeat the calculation an astronomical number of times, to have any chance of winning the block. So, it's not a mathematical "proof." It's a mechanism such that your probability of getting the next block is exactly the fraction of the hashrate that your computer is responsible for.

More resources

https://en.bitcoin.it/wiki/Protocol_documentation

Cursory analysis

The point is to make it both costly and rewarding to create the block. Proof of Work demonstrates to the network that a miner has incurred costs. The network rewards the miner by the block reward. Because it's costly to get involved in mining, miners will not do this casually. The reward mechanism creates an economic market which will trend towards equilibrium. If the block reward is worth more than the expected cost of finding a block, more miners will join in the hunt and pursue the reward. If too many miners are pursuing the reward, the expected profit will drop and miners will devote less effort.

The security is based on the following. If a transaction where Bob sends Alice some Bitcoin is confirmed in a block, say block N, no subsequent block will allow Bob to send that same Bitcoin to Charlie. So if Bob signs a transaction sending Charlie that Bitcoin, the only way the network will honor that transaction is if a miner decides to build a new block N, creating competing chains. If the latter chain becomes longer than the former, because more miners build on it, then the Bitcoin will belong to Charlie.

Now because it requires much work and some luck to create a block, miners, in absense of some other motivation, will almost always build their blocks on the longest chain. Creating a fork comes with a chance of being the losing fork, in which case the efforts are all for naught.

51% Attack

Now if a single miner, or mining pool, where to be consistently generating more than 50% of the hashrate, they would have the ability to go back to a previous block, create a fork, and then by producing blocks at a faster rate than the rest of the network, creating a longer chain which would eventually be considered the valid chain. The consensus mechanism of Bitcoin is based on the belief that this will not happen.

1.2.3 Proof of Stake

Again, the idea of Proof of Work was for distributed and possibly anonymous parties to prove they had spent some money or effort. Those that prove this, while following the protocol are rewarded. This keeps random or malicious players from messing with the network, as it will always cost something. But what if that thing that is expensive is not something consumed, like electricity, but a digital cryptocurrency or asset itself, whose ownership can be guaranteed cryptographically? This is the idea behind Proof of Stake. By only allowing users of the network who own private keys to write blocks, the network can hope to keep out malicious actors. Those with a vested interest in the network are less likely to try to cause the network harm. By paying reward for the block creators, there is then incentive to hold the coin. In practice, Proof of Stake (for example Cardano's Ouroboros protocol) is often carried out by a lottery: Each block creator is selected with probability corresponding to the fraction of the coins that they own. For example, if Alice owns 3% of the coin available, there will be a 3% chance that she is given the opportunity to create the block. Now because the percentage for each user is likely to be small (these networks hope the have millions of users), most users will not have a computer running 24 hours a day to take advantage of a slim opportunity. Instead they delegate their staking chances to another party, which is running the protocol 24 hours a day, and then are compensated. This is called Delegated Proof of Stake.

Requiring proof of an internal resource has advantages and disadvantages. An advantage is that it is not influenced by rapid changes in the availability of things like electricity or mining equipment. Another is that it doesn't produce huge externalities that may sour the brand: for example, Bitcoin consumers on order of 100 Terawatt hours per year. On the other hand, the fact that the security of the network is not separated from the functioning of the network can open open more complicated security issues. The surface area for attack is much larger and contains many more attack vectors. Because staking units can be bought, sold and transferred instantaneously, Proof of Stake lacks a certain physical security found in Proof of Work protocols. A full discussion of vulnerabilities of Proof of Stake is beyond the scope of this volume.

1.2.4 Nodes vs. miners and their control over the protocol.

Bitcoin requires several layers to function. Miners in a certain sense, are extrensic to the network, providing a service for a fee. The most basic skeleton of the network itself is formed by the nodes. The nodes are the set of computers who are all running the Bitcoin protocol, communicating with other computers on the network, rebroadcasting transactions, rebroadcasting valid blocks, and validating incoming transactions. Miners typically will be need to function at least partially as nodes, but most nodes do not function as miners. The network cannot function without both. Nodes provide connectivity, while miners provide a competitive work needed to maintain consensus. While necessary, nodes can only verify transmit information. No more nodes are needed than to maintain connectivity of the network.

Chapter 2

Probability spaces

A probability space, in the most general sense, consists of the following:

- A set Ω, called the **sample space**,
- a collection $\mathcal{F}$ of subsets of Ω, called **events,** and
- a function $P : \mathcal{F} \to [0, 1]$ which describes the probability of any event.

Example 3. *Suppose you flip three coins and observe whether each is "Heads" or "Tails". There are 8 possible outcomes which make up the sample space:*

$$\Omega = \{HHH, HHT, HTH, HTT, THH, THT, TTH, TTT\}.$$

The set of "events" is much larger, as it consists of all possible subsets of Ω (there are 64 possible subsets.) These are often described prosaically, for example, the event $\{HHH, TTT\}$ can be described as "all three coins show the same." Similary, $\{HHH, HHT, HTH, THH\}$ is the event "at least two heads."

Assuming the coin in example 3 is "fair" we can assign a probability to each event. In the case the probability of any of the sample outcomes will be the same: HHH is equality as likely as TTH. The event "at least two heads" has probability 1/2, because this involves 4 out of the 8 equally like samples, while "exactly two heads" has probability 3/8.

Example 4. *Consider sampling from the set of real numbers $0 \leq x \leq 1$ in a uniformly random way. Uniformly random, in this case means that the probability of number you sample from an interval (a, b) will be precisely $|b - a|$. Here the sample space is quite large (uncountably infinite, in fact). The event space is quite large. The easiest-to-describe and often most useful events will be collections of intervals. For example $\left\{x : \frac{1}{2} \leq x \leq \frac{2}{3}\right\}$, or $\left\{x : x < \frac{1}{3} \text{ or } x > \frac{2}{3}\right\}$. The probability for any singleton in the sample space is precisely zero - this corresponds to the probability of choosing a number from $[0, 1]$ completely at random, and having it be, say .4328. On the other hand, the probability of* *interval events will be easier to describe. The probablility of the number x being larger than $1/3$ will be $2/3$. The probability of x being in between $1/5$ and $4/5$ will be $3/5$.*

We are intentionally avoiding discussing the full set of events in this example, as it would take us well beyond the scope of this text, and would involve a long discussion of "measurability." This will not be necessary for discussion of Bitcoin.

The probability function $P : \mathcal{F} \rightarrow [0, 1]$ should satisfy some basic assumptions in order to be considered a probability function.

1. First, the probability that any random sample lies in the set that it has been randomly sampled from must be 1, so we require that

$$P(\Omega) = 1.$$

   (Notice that Ω is a subset of Ω, albeit improper, so is itself an event, that is $\Omega \in \mathcal{F}$.)

2. If two events are disjoint, the probability of the two events should add up in the obvious way. Namely

$$P(E_1 \cup E_2) = P(E_1) + P(E_2) \text{ whenever } E_1 \cap E_2 = \{\varnothing\}.$$

   In words, "the probability of E_1 or E_2 is the probability of E_1 plus the probability of E_2, whenever E_1 and E_2 do not overlap."

As an immediate corollary of these two axioms, one can derive other properties. For example, the probability of an event and its complement must sum to 1:

$$P(E) + P(E^c) = 1$$

where E^c denotes the complement of E, that is, all elements of Ω that are not in E.

By decomposing $E_1 \cup E_2$, we can derive another formula:

$$P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2).$$

Also note that because $P(E) \geq 0$ for any event, it follows that

$$P(E_2) \leq P(E_1) \text{ whenever } E_2 \subset E_1.$$

2.0.1 Countably and Uncountably infinite

A set X is **finite**, if the number of elements in X is a finite number.

A set X is called **countably infinite** if the elements of X can be put in one-to-one correspondence with the natural numbers $\mathbb{N} = \{1, 2, 3, 4, ...\}$. In other words, it must be possible to start writing down the elements of X as

$\{x_1, x_2, x_3, ...\}$ in a way that will eventually include every element of X. For example, the rational numbers are countable: One can start writing down a sequence $\{0, 1/2, 1, 1/3, 2/3, 4/3, 5/3, 2, 1/4, 3/4, ...\}$ of rational numbers in a way such that every rational number will eventually be hit. For example the rational number $q = 5087/17801239$ will be *somewhere* in the sequence if you've devised it correctly.

A set X is **uncountably infinite** if the elements of X cannot be put in one-to-one correspondence with the natural numbers. In other words, every attempt to describe X as $\{x_1, x_2, x_3, ...\}$ must necessarily omit some elements of X. The set of real numbers is uncountable.

Another important uncountable set is the set of infinite binary sequences $\mathcal{B}$, that is the set of all strings of 0's and 1's that start somewhere but do not terminate. A typical example:

$$b = 0110100100100101110010... \ .$$

The Cantor Diagonization argument confirms that this space is uncountable: Suppose that we can put all of the elements $b \in \mathcal{B}$ in order: $\mathcal{B} = \{b_1, b_2, b_3, ...\}$. We can prove by contradiction that this is impossible: Consider the "diagonal" sequence consisting of the first digit of b_1, the second digit of b_2, etc. By toggling the digits, we can construct a new sequence b^* which cannot have possibly occurred in the ordering $\{b_1, b_2, b_3, ...\}$. If b^* were found in this ordering, it would be at position n, for some n, that is $b^* = b_n$. But we have constructed b^* to have the opposite nth digit as b_n. So we conclude that the set $\mathcal{B}$ is uncountable.

Countable sets or countable events are easier to deal with for probability theory. By putting the samples in an order one can cycle through all of them and make statements such as

$$P(E) = \sum_{\omega \in E} P(\omega_i).$$

For an uncountable set, such a sum would not make sense. When dealing with uncountable sets, it's often useful to partition them into countably many subsets.

2.1 Partitions

For many purposes we will need to divide a probability space into either finite or countably infinite events, such that

$$\bigcup_i E_i = \Omega$$
$$E_i \cap E_j = \{\varnothing\} \text{ for all } i, j.$$

For Example 3, one choice of partition is according to the first flip, that is, divide into two events: E_1 ="first flip is heads" and E_2 = "first flip is tails."

We may find it convenient to relax the definition slightly, up to events with probability zero. A more general definition would be

$$P\left(\bigcup_i E_i\right) = 1$$
$$P\left(E_i \cap E_j\right) = 0 \text{ for all } i, j$$

which means, in other words, any random outcome ω with probability 1 is found in at least one subset of the partition E_i and will be in more than one subset with probability 0.

For example, for integers $i \geq 1$, let

$$E_i = \left\{x : \frac{1}{i+1} \leq x \leq \frac{1}{i}\right\}.$$

This partitions the unit interval $[0, 1]$,

$$E_1 \cup E_2 \cup E_3 ... = \left[\frac{1}{2}, 1\right] \cup \left[\frac{1}{3}, \frac{1}{2}\right] \cup \left[\frac{1}{4}, \frac{1}{3}\right] ...$$

The endpoint $\{0\}$ does not occurs, and the points $\{1/n\}$ occur more than once, however, the probability of any of these being sampled is zero. If a countable set of outcomes each has probability zero, then the event consisting of the countable union is also zero.

2.1.1 Conditional Probability

Oftentimes we will be given partial information. We may know that an event has occurred, and will want to know if this gives us a better idea of the probability of other events. We define the conditional probability:

$$P(E \mid F) = \frac{P(E \cap F)}{P(F)}$$

which is read as "the probability of E, given F". We know the event F has occurred and we use this information to determine the probability of event E occurring. In Example 3. Let E be "all flips are Heads" and F be "at least two heads." We have

$$P(\text{all three heads} \mid \text{at least two heads}) = \frac{P(E \cap F)}{P(F)}$$
$$= \frac{1/8}{3/8} = \frac{1}{3}.$$

In other words, if you have the information that at least two heads have been flipped (you don't know which ones or the order) you now conclude there is a 1/3 probability that all three are heads.

We say that two events E, F are independent if

$$P(E)P(F) = P(E \cap F). \tag{2.1.1}$$

It follows that for independent events

$$P(E \mid F) = \frac{P(E)P(F)}{P(F)} = P(E)$$

or in other words, knowing that F has occurred does not change the likelihood that E has occurred, justifying the definition.

2.2 Random Variables

A random variable is essentially a function on the space Ω. That is, if $f : \Omega \to \mathbb{R}$ determines a real number for each element in the sample space, we say f is a random variable. The value of $f(\omega)$ depends on the randomly sampled event, so is random in this sense.
(A mathematican may find this definition somewhat strange. For statisticians, the random variable is often a determinist function of a dependent variable that was to be chosen at random, so the function itself feels neither "random" nor "variable," words which would well-describe the argument of the function.)

In Example 3, one could let f be "the number of Heads". This is a random variable: For each outcome in the sample space, one can determine the number of heads.

In Example 4, one could define a random variable $f(x) = x^2$. Given an $x \in [0, 1]$ chosen at random, square it.

2.2.1 Expectation

A real-valued random variable will have an expectation, also known as "expected value," "mean," "average," or "weighted average." This is denoted $\mathbb{E}[f]$. One can think of $\mathbb{E}[f]$ as the average value after many trials are repeated.

For a finite probability space, the definition is simple

$$\mathbb{E}[f] = \sum_{\omega_i \in \Omega} f(\omega_i)P(\omega_i). \tag{2.2.1}$$

A very similar definition holds for countably infinite spaces, but one has to make the caveat that the sum converges, (cf. the St. Petersburg Paradox.) For example the "expected number of heads" in Example 3 will be

$$\frac{1}{8}\{3+2+2+2+1+1+1+0\} = \frac{12}{8}.$$

For infinite spaces, the expectation can often be expressed as an integral: For example to compute the expected value of x^2 we compute

$$\mathbb{E}[x^2] = \int_0^1 x^2 dx = \frac{1}{3}.$$

(There's a lot here we are deliberately not unpacking. It would require several chapters.)

The **variance** of random variable is defined as

$$Var(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]. \tag{2.2.2}$$

In other words, variance is the expected value of the difference between the value and it's expected value, squared. If a function is usually close its mean, it will have small variance. The **covariance** for two random variables is

$$Cov(X,Y) = \mathbb{E}[(X - \mathbb{E}\left[X\right])(Y - \mathbb{E}\left[Y\right])] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y].$$

If X and Y tend to be above their means at the same times as one another, the covariance will be positive. The variance satisfies the formula

$$Var(X + Y) = Var(X) + Var(Y) + Cov(X\,, Y).$$

Variables X and Y are considered **independent** if information about the value of X does not give you any information about the value of Y. In other words, any event involving knowledge of X sheds no new light on Y. For example,

$$P\left(X \geq 3 \mid Y \leq 9\right) = P\left(X \geq 3\right).$$

This implies, and is a stronger condition than $Cov(X,Y) = 0$. For a larger set of variables, say $X_1, X_2, .., X_n$, we say these are independent if information about any combination of them does not give us information about variables not included in that combination.

Importantly for our concerns, whenever variables $X_1, X_2, ...X_n$ are independent we have

$$Var\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} Var\left(X_i\right). \tag{2.2.3}$$

Note that the operation of computing the expectation is linear in the random variable. Given two random variables f and g

$$\begin{aligned}
\mathbb{E}[f+g] &= \sum_{\omega_i \in \Omega} \left(f(\omega_i) + g(\omega_i)\right) P(\omega_i) \\
&= \sum_{\omega_i \in \Omega} \left(f(\omega_i)P(\omega_i) + g(\omega_i)P(\omega_i)\right) \\
&= \mathbb{E}[f] + \mathbb{E}[g].
\end{aligned}$$

Note that this is true even if f and g are not independent.

2.2.2 Conditional expectation

Condition expectation is a convenient computational tool that will be frequently used in the upcoming sections. This allows us to split the computations into pieces that will often be more manageable. If we have a nice partition of the sample space, we can compute the conditional expectations over each of the events in the partition, and then sum these up appropriately.

This involves the quantity $\mathbb{E}[f \mid E]$, or "the conditional expectation of f, given event E." In other words, if we restrict from the whole set Ω to the event E what is our expectation for f? For finite or countable probability spaces, we use (similar to (2.2.1))

$$\begin{aligned}\mathbb{E}[f \mid E_i] &= \sum_{\omega_i \in \Omega} f(\omega_i) P(\omega_i \mid E) \\ &= \sum_{\omega_i \in E} f(\omega_i) \frac{P(\omega_i)}{P(E)}.\end{aligned} \tag{2.2.4}$$

Here we are using the fact that for each singleton ω_i we have

$$\begin{aligned}&P(\omega_i \mid E) = \frac{P(\omega_i)}{P(E)} \text{ if } \omega_i \in E \\ &P(\omega_i \mid E) = 0 \text{ if } \omega_i \notin E.\end{aligned}$$

This allows us to use a partition to compute expectation, via the following formula:

$$\mathbb{E}[f] = \sum_i \mathbb{E}[f \mid E_i] P(E_i). \tag{2.2.5}$$

In other words, the weighted average is simply the weighted average of a set of weighted averages. The proof of this formula is easy; given (2.2.4)

$$\begin{aligned}\sum_i \mathbb{E}[f \mid E_i] P(E_i) &= \sum_{E_i} \left(\sum_{\omega_j \in \Omega} f(\omega_j) P(\omega_j \mid E_i) \right) P(E_i) \\ &= \sum_{E_i} \left(\sum_{\omega_j \in E_i} f(\omega_j) \frac{P(\omega_j)}{P(E_i)} \right) P(E_i) \\ &= \sum_{\omega_j \in \Omega} f(\omega_j) P(\omega_j) = \mathbb{E}[f].\end{aligned}$$

Returning again to Example 3, and letting f be the number of Heads we see; if we partition into E_1 = "first flip is Heads" and E_2 = "first flip is Tails" we have

$$\begin{aligned}\mathbb{E}[f] &= \mathbb{E}[f \mid E_1] P(E_1) + \mathbb{E}[f \mid E_2] P(E_2) \\ &= \frac{1}{2}\mathbb{E}[f \mid E_1] + \frac{1}{2}\mathbb{E}[f \mid E_2].\end{aligned}$$

Intuitively (can be verified by writing out the full sum)

$$\mathbb{E}[f \mid E_1] = 2$$

and

$$\mathbb{E}\left[f \mid E_2\right] = 1$$

so not surprisingly, this recovers

$$\mathbb{E}[f] = \frac{1}{2}\left(2+1\right) = \frac{3}{2}.$$

One useful partition is according to the value of the random variable itself. For example $E_c = \{\omega \mid f(\omega) = c\}$ defines an event. If the random variable f has a finite or countable range, the expected value can be written as

$$\mathbb{E}[f] = \sum_{c \in range(f)} c \times P(f = c).$$

In Example 3 this gives us

$$\begin{aligned}\mathbb{E}[\text{number of heads}] &= \sum_{c \in \{0,1,2,3\}} c \times P(\text{number of heads } = c)\\ &= 0 \times \frac{1}{8} + 1 \times \frac{3}{8} + 2 \times \frac{3}{8} + 3 \times \frac{1}{8}\\ &= \frac{12}{8}.\end{aligned}$$

2.3 Bernoulli Processes, Bernoulli Schemes

Most of the mining games will be best presented in terms of Bernoulli processes. A **Bernoulli process** is a finite or infinite sequence of independent binary-valued random variables $\{X_1, X_2, X_3, ..\}$ such that each $X_i \in \{0,1\}$ and for all values of i we have

$$\begin{aligned}P\left\{X_i = 1\right\} &= p\\ P\left\{X_i = 0\right\} &= 1-p\end{aligned}$$

We emphasize that the probability doesn't depend on i.

The probability space involved is the space of binary sequences:

$$\mathcal{B} = \left\{\text{Set of all binary sequences}\right\}.$$

Note that with this description of the space, the random variables X_i desribed above are simply the ith entry in the sequence.

For Bernoulli processes, the probability space is determined by a parameter p describing the probability that each given trial will be 0 or 1. This allows us to compute the probability of a rich set of events. For example:

- The probability that any given $X_i = 1$ is p.

- The probability that the first two outcomes are both 0 is $(1-p)^2$. Since $P(X_1 = 0) = (1-p)$ and $P(X_2 = 0) = (1-p)$ and these two events are independent, we conclude

$$P(X_1 = 0 \text{ and } X_2 = 0) = (1-p)\,(1-p)$$

  using (2.1.1).

- The probability that $X_1 \neq X_2$ is $2p(1-p)$. This is the sum of probabilities of two disjoint events, each with probability $p(1-p)$.

Example 5. *(Random Walker). A drunkardess lurches forwards or backward in a series of steps. The probability that she steps forwards is p, the probability she steps backwards is $1-p$. After 10 steps, what can we say about where she is?*

When we are only interested in what happens after a fixed (10) number of step we don't need to use the full infinite space $\mathcal{B}$. We can deal with $\mathcal{B}^{(10)}$, the set of binary sequences of length 10. This set has $2^{10} = 1024$ elements. The probability that the walker has moved ten steps ahead is p^{10}, ten steps backward $(1-p)^{10}$, these probabilities are easy to compute because for each there is exactly one sequence in this event. To compute the probability that the drunkard has moved 8 steps ahead, observe that this involves 10 different possibilities. The walker must take nine steps forward and one step back, but because the order in which this occurs does not matter, there are 10 different steps that could have been taken backward. The probability for each individual path is $p^9(1-p)$. So

$$P(\text{walker is 8 steps ahead}) = 10 \times p^9(1-p).$$

The probability that the walker is 6 steps ahead is more interesting. We know that the walker must have taken two steps backwards, and we don't care which of the 10 steps these were. However, we do have to be careful not to count the possibilities twice. This brings us to combinatorics. The notation

$$\binom{10}{2}$$

is read "10 choose 2" and denotes the number of distinct ways to choose 2 out of a set of 10. We can think of this as follows. There are 10 possibilities to pick the first one. Once the first one has been chosen there are 9 possibilities to choose the second one. This gives us 90 ways to choose a distinct ordered pair from 10 possibilities. However, choosing 4 then 8 is no different from choosing 8 than 4, at least for our purposes, which means the quantity 90 has over-counted everything by a factor of 2. We conclude

$$\binom{10}{2} = \frac{10 * 9}{2} = 45.$$

In general the formula determining ways to select k elements from a set of n is given by.

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}.$$

Using this reasoning, we see that

$$\begin{aligned} P\left(\text{walker is 6 steps ahead}\right) &= \binom{10}{2} p^8(1-p)^2 \\ P\left(\text{walker is 4 steps ahead}\right) &= \binom{10}{3} p^7(1-p)^3 \\ &\dots \\ P\left(\text{walker is 4 steps behind}\right) &= \binom{10}{7} p^3(1-p)^7 \\ &\text{etc.} \end{aligned}$$

More generally, a **Bernoulli scheme** is similar, but with the possibility of more than two outcomes each trail.

Given a finite collection of mining pools, the sequence of pools who attain the next target hash is a Bernoulli scheme. The random variable X_i will be the answer to the question; "which pool produced the ith hash meeting the target." (Random variables need not always be real-valued.) From the perspective of a single pool, the sequence of "my pool" or "not my pool " is a Bernoulli process. Note that

$$\begin{aligned} P\left\{X_i = \text{“my pool”}\right\} &= p \\ P\left\{X_i = \text{“not my pool”}\right\} &= 1-p \end{aligned}$$

whenever the pool controls a fraction p of the hashrate.

2.3.1 The binomial distribution formula and de Moivre–Laplace theorem

When $X_i \in \{0,1\}$ we can try to compute statistics for the sum of the first n outcomes. Define

$$S_n = X_1 + X_2 + \dots + X_n.$$

First observe that

$$\mathbb{E}\left[X_i\right] = p$$

and by the linearity of expectation,

$$\mathbb{E}\left[S_n\right] = np.$$

Now for a particular k we use the analysis in the random walker example to compute $P(S_n = k)$: In order for S_n to be k there must have been k instances among the first n where $X_i = 1$. There are $\binom{n}{k}$ chooses of these, each has probability $p^k(1-p)^{n-k}$. Thus

$$P(S_n = k) = \binom{n}{k} p^k (1-p)^{n-k}.$$

For n not too large, these numbers can be computed by hand (or by computer, via the hand.) The function

$$\phi(k, n, p) = \binom{n}{k} p^k (1-p)^{n-k}$$

is called the **probability mass function**. As n becomes large, we will be less interested in the small possibility that S_n is exactly equal to a particular k and more interested in when S_n is below or above certain thresholds. This requires the **cumulative distribution function.** For a random variable f, the cumulative distribution function is defined by

$$F(s) = P(f \leq s).$$

Necessarily, we will have

$$\lim_{s \to -\infty} F(s) = 0$$

and

$$\lim_{s \to \infty} F(s) = 1.$$

For the particular case of binomial distributions, we have for $k \in \mathbb{N}$

$$\begin{aligned} F(k; n, p) &= P(S_n \leq k) \\ &= \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}. \end{aligned}$$

This can be somewhat involved to compute by hand. It can also be expressed via the incomplete beta function I as

$$\begin{aligned} F(k; n, p) &= I_{1-p}(n-k, k+1) \\ &= (n-k)\binom{n}{k} \int_0^{1-p} t^{n-k-1}(1-t)^k dt. \end{aligned}$$

(We're not expecting the average reader to know what the beta function is, why it exists, and we won't offer a proof of this statement. This is just a fun fact.) This expression may or may not be easier to compute. We will revisit this in more detail in section 5.1.

An approximation formula is given by thede Moivre–Laplace theorem:

$$\phi(k, n, p) \approx \frac{1}{\sqrt{2\pi n p(1-p)}} e^{-(k-np)^2/(2np(1-p))}$$

which is a closely akin to the Central Limit Theorem, which applied to S_n states that as $n \to \infty$, the random variable

$$\tilde{S}_n = \frac{(S_n - np)}{\sqrt{np(1-p)}} \tag{2.3.1}$$

converges to a random variable with standard normal distribution. Thus for very large values of n, one may use the cumulative distribution function for the standard normal distribution:

$$\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-s^2/2} ds.$$

For example, if $p = 0.4$ and $n = 10,000,000$ what is the probability that $S_n \leq 4,001,000$?

Using (2.3.1) the event that $S_n \geq 4,100,000$ is the same as the event that

$$\tilde{S}_n \leq \frac{4,001,000 - 4,000,000}{\sqrt{10,000,000 * (0.4)(0.6)}} = 0.645\,50.$$

If $\tilde{S}_n$ is a normally distributed random variable, we have

$$\begin{aligned} P(\tilde{S}_n \leq 0.645\,50) &= \Phi\,(0.645\,50) \\ &\approx 0.740\,70. \end{aligned}$$

For a statement and proof, see [SS2011, pg. 197].

2.3.2 Probability-preserving isomorphisms and comparison of expectations

Occasionally we will need to consider very similar spaces with identical properties. Clearly the space of Bernoulli trials with $p = 1/3$ where the outcomes are $\{0, 1\}$ must be equivalent in some way to the space of Bernoulli trials with $p = 1/3$ and the outcomes are $\{\mathrm{A}, \mathrm{B}\}$. We can make this explicit: Define the map

$$T : \mathcal{B}_{\{0,1\}} \to \mathcal{B}_{\{\mathrm{A},\mathrm{B}\}}$$

via

$$T(00101...) = \mathrm{AABAB...}$$

in the obvious way. Then you can check that T is probability preserving in the sense that

$$P\{x \in E\} = P\{T(x) \in T(E)\}.$$

This will be useful to us via the following: Consider a mining pool with p hashrate. Suppose in one universe they decide to mine a given transaction, suppose in another universe they decide to not mine that same transaction. Moving forward, the sequence of Bernoulli trials determined by whether or not this pool obtained the block will look different: because the hashing function behaves chaotically, the fact that the pool obtained the next block in former situation is not going to guarantee the pool obtained the next block in the latter.

Many of our computations will involve trying to maximize the expected value of a function, given a decision between two or more options. For example, letting R be the rewards a given pool earns over the next 1000 blocks, we would like to compute $\mathbb{E}(R)$ in the case where we pursue a certain strategy and compare it to the case where we don't pursue a certain strategy. Naively, the Bernoulli sequence describing who wins the blocks are not the same. However, they are isomorphic: The probability that an event E occurs on one is equal to the probability that a corresponding event $\tilde{E}$ occurs in the other. This allows us to compute the difference of the two $\mathbb{E}(R)$ at the same time, for example using the same partition, without worrying about the fact that these are technically different probability spaces.

Left hand right hand game.

We illustrate this with a contrived example. Suppose you have to chose between two games. Game 1 is the following: You flip a coin with your left hand. You win \$10 when you flip heads, you lose \$1 for every tails and then game is over when you flip heads for the first time. Game 2 you flip with your right hand. You win \$15 for every heads and you lose \$1 for every tails, and once you flip a heads the game is over. Now intuitively the right-hand game is better, but we ignore that for a minute. Computing the expectation for either game is possible. A naive approach in comparing the two games is to compute the expectation for one and then the expectation for the other, and compare them side-by-side. However, this computation may end up more complicated. Perhaps you're not interested in knowing exactly how much you will win: You may be more motivated to know how much you would be willing to pay to upgrade from Game 1 to Game 2. It will be an easier computation to compare the difference of these two games, using the fact that both games occur on the same two isomorphic underlying probability spaces. If you perform a series of coin-flips with my left hand, it will produce a Bernoulli process with identical distribution as if you had flipped the coin with my right hand, even though the sequence sampled will not literally the same.

We define a function f as follows: f is the money won by playing the left-hand game, subtracted from the money won playing the right-hand game. The game will terminate with probability 1. We may partition the space by defining E_i to be the event that the first H is flipped on the ith flip. Then we compute

$$\mathbb{E}[f] = \sum_{i=1}^{\infty} \mathbb{E}\left[f \mid E_i\right] P(E_i)$$

Now we notice the follow: For event E_i we have the left hand game wins $10 - i$, while the right hand game wins $15 - i$. The difference is thus the constant number 5. So we have

$$\mathbb{E}[f] = \sum_{i=1}^{\infty} 5P(E_i) = 5\sum_{i=1}^{\infty} P(E_i) = 5.$$

This is much easier than computing

$$\begin{aligned}&\mathbb{E}[f_{RH}] - \mathbb{E}[f_{LH}] \\ &= \sum_{i=1}^{\infty}(15-i)\left(\frac{1}{2}\right)^i - \sum_{i=1}^{\infty}(10-i)\left(\frac{1}{2}\right)^i.\end{aligned}$$

This may appear contrived and pedantic now, but this framework will make computations about whether or not to pursue a reorg much easier.

Justification. Suppose we have a probability preserving isomorphism

$$T : \Omega \to \tilde{\Omega}.$$

And we have a partitions

$$\begin{aligned}&\{E_i\} \\ &\left\{\tilde{E}_i = T(E_i)\right\}\end{aligned}$$

of both such that

$$P_{\Omega}(E_i) = P_{\tilde{\Omega}}(T(E_i)).$$

Certainly,

$$\sum P(E_i) = 1 = \sum P(\tilde{E}_i).$$

We have

$$\begin{aligned}E_{\Omega}(f) &= \sum P(E_i)E(f \mid E_i) \\ E_{\tilde{\Omega}}(\tilde{f}) &= \sum P(\tilde{E}_i)E(\tilde{f} \mid \tilde{E}_i)\end{aligned}$$

so

$$\begin{aligned}E_{\Omega}(f) - E_{\tilde{\Omega}}(\tilde{f}) &= \sum P(E_i)E(f \mid E_i) - \sum P(\tilde{E}_i)E(\tilde{f} \mid \tilde{E}_i) \\ &= \sum P(E_i)\left\{E(f \mid E_i) - E(\tilde{f} \mid \tilde{E}_i)\right\}\end{aligned}$$

because the probabilities are the same $P(E_i) = P(\tilde{E}_i)$. Now if I can determine f and $\tilde{f}$ from E_i and the corresponding $\tilde{E}_i$ I can use the simplified formula

$$E_{\Omega}(f) - E_{\tilde{\Omega}}(\tilde{f}) = \sum P(E_i)E(f - \tilde{f} \mid E_i).$$

Chapter 3

Game theory basics

3.1 Single round games

Here we begin with some very simple examples. We will then discuss and formalize some definitions. In the examples in this section players make moves simultaneously in a single round, without knowledge of the other player's strategy.

Example 6. *Rock-Paper-Scissors: Player 1 and Player 2 simultaneously choose Rock, Paper or Scissors. The rule is Rock beats Scissors, Scissors beats Paper and Paper beats Rock. This can be summarized in the following table.*

Player 2 choice / *Player 1 choice*	*Rock*	*Paper*	*Scissors*
Rock	*Tie*	*Player 2 wins*	*Player 1 wins*
Paper	*Player 1 wins*	*Tie*	*Player 2 wins*
Scissors	*Player 2 wins*	*Player 1 wins*	*Tie*

Example 7. *Prisoner's Dilemma. Two partners in a bank robbery are apprehended while jaywalking and brought into questioning separately. Each are given the following choice: 1) Confess to the major crime and implicate their associate, or 2) Refuse to implicate their associate but face charges for jaywalking.*
The outcomes are summarized as follows.

Player 2 choice / *Player 1 choice*	*Stay Silent*	*Accuse Partner*
Stay Silent	*Both pay fines for jaywalking*	*Player 2 goes free, Player 1 to jail for both bank robbery and fined for jay-walking*
Accuse Partner	*Player 2 to jail for both bank robbery and for jaywalking, Player 1 goes free,*	*Both go to jail for bank robbery*

Example 8. *Splitting $4. Two players are supposed to split $4. They each write down a number $1, $2 or $3. If the sum of the numbers is no more than $4, each player gets the number they wrote down. Otherwise, each gets nothing. This can be summarized in the following table.*

Player 2 choice / *Player 1 choice*	*1*	*2*	*3*
1	*Each gets $1*	*Player 1 gets $1, Player 2 gets $2*	*Player 1 gets $1, Player 2 gets $3*
2	*Player 1 gets $2, Player 2 gets $1*	*Each gets $2*	*Each gets nothing*
3	*Player 1 gets $3, Player 2 gets $1*	*Each gets nothing*	*Each gets nothing*

Payoff Tables

In the sequel, it will be convenient to use a **payoff table** to describe two-person games. The payoff table for the above example (Example 8) looks like

	1	2	3
1	(1,1)	(1,2)	(1,3)
2	(2,1)	(2,2)	(0,0)
3	(3,1)	(0,0)	(0,0)

In the first example (Example 6) the payoff table would look like

	Rock	Paper	Scissors
Rock	(T,T)	(L,W)	(W,L)
Paper	(W,L)	(T,T)	(L,W)
Scissors	(L,W)	(W,L)	(T,T)

In general, when we see (a, b) in a payoff table it means that Player 1 wins proceeds a while Player 2 wins proceeds b.

Summary

As we see, games consist of the following.

1. A set of players.
2. A set of moves by each player.
3. The outcomes of the combinations of moves.
4. The preferences of the players for the different outcomes.

3.1.1 Preferences and utility function

In the above examples, the preferences are clear. The Rock-Paper-Scissors players prefer Win to Tie, and prefer Tie to Loss. (We don't know how strongly they prefer this - the players could be in an RPS league in which Ties is only slightly preferable to Loss.)

In analysing games, usually what we need to know is the preference between different outcomes. For example, a Rock-Paper-Scissors player, if given a choice, will choose to Win rather than Lose. Similarly, in games involving money, it's evident that players prefer more money rather than less. It will be useful to try to quantify this, with a utility function, U, describing how "good" that outcome is for that player. This function assigns numeric values to different outcomes and will be useful in determining players' strategies. For example, in Rock-Paper-Scissors it may be useful to assign

$$\begin{aligned} U(\text{Win}) &= 2 \\ U(\text{Loss}) &= 0 \\ U(\text{Tie}) &= 1. \end{aligned} \tag{3.1.1}$$

Note that if we were to use a different utility function, for example

$$\begin{aligned} U(\text{Win}) &= 1 \\ U(\text{Loss}) &= -1 \\ U(\text{Tie}) &= 0 \end{aligned}$$

or even

$$\begin{aligned} U(\text{Win}) &= 10 \\ U(\text{Loss}) &= 0 \\ U(\text{Tie}) &= 1 \end{aligned} \tag{3.1.2}$$

we get the same ordering.

Whenever the game is deterministic (no probability involved) any utility function that preserves the ordering will result in the same analysis.

If we have fixed a utility function, we can then create a payoff table, for example, if we use the function (3.1.1) we get the following payoff table for RPS:

	Rock	Paper	Scissors
Rock	$(1,1)$	$(0,2)$	$(2,0)$
Paper	$(2,0)$	$(1,1)$	$(0,2)$
Scissors	$(0,2)$	$(2,0)$	$(1,1)$

For the Prisoner's Dilemma game, we could try to quantify it by saying that going to jail for bank robbery is 10000 times as bad as being fined for jaywalking. The payoff function would look like:

Player 2 choice / Player 1 choice	Stay Silent	Accuse Partner
Stay Silent	(-1,-1)	(-10,001,0)
Accuse Partner	(0,-10,001)	(-10,000,-10,000)

However, from a purely preferential perspective, this is equivalent to the payoff table:

Player 2 choice / Player 1 choice	Stay Silent	Accuse Partner
Stay Silent	(3,3)	(0,5)
Accuse Partner	(5,0)	(1,1)

All of the preferential orderings in the above two tables are preserved, so there is a sense in which these games are equivalent.

Matrix Games

Matrix games are a standard point of exploration for two-player game theory. These are zero-sum games, meaning that the loss of one player is equal to the gain of the other player. The payoffs are easy to represent by writing a single number representing the reward to Player 1. For example Rock-Paper-Scissors can be rewritten as

	Rock	Paper	Scissors
Rock	0	-1	1
Paper	1	0	-1
Scissors	-1	1	0

These can also be described as "row/column" games: Player 1 chooses a Row , Player 2 chooses a column, and Player 2 then gives Player 1 the value specified in the matrix, or receives that number if the entry is negative.

Example 9. *Matrix Game*

$$\begin{pmatrix} 3 & 6 & 1 & -4 \\ -3 & 0 & -4 & 7 \\ -1 & 10 & -3 & -1 \\ 5 & -6 & 1 & 5 \end{pmatrix}$$

As in the games we've discussed so far, it is assumed that both players are playing their strategy without knowing the other player's strategy. Which row should player 1 choose?

Robinson Crusoe Games.

Some games don't involve other people, but are instructive to think about. A single player desires to maximize his own utility and doesn't have to think about what other players are doing. This becomes simply a question of optimization. For example, an individual with $25 in his pocket at a street fair can decide whether to buy a beer and a bratwurst or an elephant ear and mango smoothie, or perhaps other combinations that may be available. This maybe be complicated by the possibility of a longer line at the beer cart. He has some preference between the combinations of possibilities, and if acting rationally, will choose the option he prefers the most. At the most basic form, this is simple economics. It will get more interesting when probability is involved.

Example 10. *Consider the TV game show "Deal or No Deal." A player chooses one of a handful of briefcases, and keeps this briefcase closed. One briefcase contains $1 Million, all other briefcases contain lesser amounts. The player then opens several other cases. As more information is revealed, the "Bank" (which for these purposes should be considered a non-playable-character) offers the player an amount for the briefcase in the player's hands. For example, it may be towards the end of the game that there are two unopened briefcases, one containing $1 Million and one containing $1000. The player holds one but does not know which one. The player may choose to open it, and wins the contents, whatever they are. The bank offers him $400,000. Should he take it?*

More reasonable utility functions and probability

The ability of a utility function to describe utility accurately becomes important when the game involves probability. In the Deal or No Deal example above, the player has two options, and must decide a preference between the two. The options are 1) Open the case he is holding, or 2) Take the $400,000. Now to properly assess this it makes sense to use probability theory. If there is a 50-50 chance of winning $1 Million, so thinking of the winnings as a random variable, the expected value is

$$\frac{1}{2} * \$1,000,000 + \frac{1}{2} * \$1,000 = \$500,500$$

which is clearly larger than \$400,000. However, the player should not be averaging over the expectation of the total dollar amount; but rather the expected utility. The principle at play is that the utility of money might be a concave function in terms of the nominal amount. In other words the utility of \$1,000,000 might not be 1000 times the utility of \$1000. This makes sense. Note that this is not a "gambling premium." If you choose to take \$400,000 over the 50-50 chance of \$1,000,000 you are probably in a situation where \$400,000 will improve your life quite significantly, an amount that \$1,000,000 would not double.

In this text, we will usually assume that the players have rather large banks, and so the concavity of the utility of money is not felt. Fortunately most of the games we discuss will denote currency valued in one fixed denomination, so we will simply be able to use that value as the payoff function in our games.

3.1.2 N-players games and more terminology

Many games involve more than two players. It won't always be possible to draw such nice tables; in a three player game one would need a three dimensional array. In general, we set up the following notation. Player p_i has a choice of moves, or strategies, which are given by a set S_i. The Cartesian product space

$$S_1 \times S_2 \times ... \times S_n$$

is the set of all possible choices of strategies for all players. For example, in RPS there are nine possible pairs of moves by the players.

$$S_1 \times S_2 = \{(R,R),(R,P),(R,S),(P,R),(P,P),(P,S),(S,R),(S,P),(S,S)\}$$

There will be an outcome function, which we call Y, which takes any n-tuple of strategies and gives an outcome in the outcome set O, which can be written

$$Y : S_1 \times S_2 \times ... \times S_n \to O$$

For RPS we have the outcome set:

$$O = \{\text{Player 1 wins, Player 2 wins, Tie}\}.$$

When we have a utility function, we can then map the outcomes to utility vectors for each player, namely

$$U : O \to \mathbb{R}^n$$

that is, for each possible outcome, an ordered tuple of number describes the utility to each player.

$$\begin{aligned} U\,(\text{Player 1 wins}) &= (2,0) \\ U\,(\text{Player 2 wins}) &= (0,2) \\ U\,(\text{Tie}) &= (1,1)\,. \end{aligned}$$

Often, we will simply short circuit the outcome stage and simply write $\tilde{U}$ as a function on the strategy space, that is

$$\tilde{U} = U \circ Y : S_1 \times ... \times S_n \to \mathbb{R}^n.$$

For example

$$\tilde{U}(\text{Rock, Paper}) = (2,0)\,.$$

Because this is a vector-valued function, we can write the utility to each player as

$$\begin{aligned} &\tilde{U}_1 : S_1 \times ... \times S_n \to \mathbb{R} \\ &\tilde{U}_2 : S_1 \times ... \times S_n \to \mathbb{R} \\ &\quad \textit{etc.} \end{aligned}$$

For 2-player games with finitely many moves, this is general nonsense that is just as easily encoded by a payoff table. However, when the moves lie on a continuum or there are many players it will make more sense to describe the game using these functions.

So far, we have described **deterministic** games, meaning the outcome is completely determined by knowing the players' strategies, by say, reading it off the table or using the function Y.

State of a game

A game can be played in multiple stages, and the later stages can be dependent on the state of the game. This will be described more in section 3.3.

A good general purpose word that we will use is the **state**. In computer science, the state is often used to describe the current value of all of variables. In a blockchain setting, the state may involve any current variables that are relevant, such as all of the transactions in the longest chain, or other signed and valid transactions that may be competing with transactions that have been confirmed. For a game, the state can be thought of as describing all information needed to reconstruct the game at that point in time, say, after a postponement due to rain. For example, in baseball, the state will include the score, the count, which runners are on base, the current lineup, etc. Typically the state is a description of any variables that can change during the course of a game or software program.

A players' strategy will be a complete contingent plan that specifies their behavior for each state in the game, describing each action a player would take at every possible decision point.

3.1.3 Dominant Strategies

A player's strategy, in this setting is simply their move. If the game has multiple stages, the strategy involves a response to any possible game state. Let s, $t \in S_i$ be two strategies in Player i's strategy space. We say that strategy s **weakly dominates** strategy t, if the strategy s results in at least the utility of t regardless of other players' strategies, and is strictly superior in at least one case. In other words, if s and t are strategies for Player 1,

$$\tilde{U}_1(s, s_2, .., s_n) \geq \tilde{U}_1(t, s_2, .., s_n)$$

for each fixed $(s_2, ..., s_n)$ and for at least some $(s_2^*, .., s_n^*)$

$$\tilde{U}_1(s, s_2^*, .., s_n^*) > \tilde{U}_1(t, s_2^*, .., s_n^*).$$

This only makes sense in terms of Players 1's strategy set and payoff function, so it is necessarily regarding only one player's perspective at a time.

We say that strategy s **strictly dominates** strategy t, if the strategy s results in superior utility to that of t regardless of any combination of other players' strategies, and the converse is not also true, namely

$$\tilde{U}_1(s, s_2, .., s_n) > \tilde{U}_1(t, s_2, .., s_n)$$

for each fixed $(s_2, ..., s_n)$.

A (strictly, weakly) **dominant strategy** is a strategy that (strictly, weakly) dominates all other available strategies.

Examples

- For RPS (Example 6), no player has a dominant strategy. If Player 1 chooses Rock, this will be superior or inferior to the choice of Paper, depending on the move of Player 2. Because it must be superior regardless of Player 2's move, it cannot be a dominant strategy.

- For Prisoner's Dilemma (Example 7), each player has a strictly dominant strategy: Accuse the other. Regardless of what Player 2 does, Player 1 can always improve their situation by accusing Player 2. Similarly for Player 2.

- In Splitting \$4 (Example 8), there is no dominant strategy. If Player 1 writes down \$1, certainly he could have made better choice in writing down \$2, in the situation where Player 2 wrote down \$1 or \$2. If Player 2 writes down \$2, they could have done better by writing down \$3 in the unlikely situation that player 2 wrote down \$1. If player 1 writes down \$3, this will only be a good choice if Player 2 writes down \$1, as all other choice will results in winning nothing.

Stag Hunt is another classic example. As it has some variations, we specify the following instance for our discussion.

Example 11. *Stag Hunt. Two hunters set out hunting. They must cooperate to catch a stag, but either can catch a hare on their own. A stag is much preferable to a hare. An example payoff table describing these preferences is*

Hunter 2 / *Hunter 1*	*Hunt Stag*	*Hunt Hare*
Hunt Stag	*(5,5)*	*(0,3)*
Hunt Hare	*(3,0)*	*(2,2)*

That is, if both decide to hunt hare, they both get hares, but if one decides to hunt hare while the other hopes to hunt stag, the latter gets nothing, while the hare hunter may bring in slightly more hare.

There is no dominant strategy: If Player 1 hunts hare, they are missing out if Player 2 has decided to hunt stag. If Player 1 hunts stag, they are left without if Player 2 decides to hunt hare.

3.1.4 Nash equilibria

Clearly, if all players have a dominant strategy, they will play this. However, this usually isn't the case. Instead players can look for a strategy that is best, provided they have some idea about the moves by the other players. A Nash equilibrium is a n-tuple of strategies that maximizes each player's outcome among their possibilities, provided that no other players deviate. In math-y notation, this is the following : A Nash equilibrium is an n-tuple of strategies (simply a pair in the case of two player games) $(s_1, ..., s_n)$ such that

$$\begin{aligned} \tilde{U}_1(s_1, s_2, ..., s_n) &\geq \tilde{U}_1(t, s_2, ..., s_n) \quad \forall t \in S_1 \\ \tilde{U}_2(s_1, s_2, ..., s_n) &\geq \tilde{U}_2(s_1, t, ..., s_n) \quad \forall t \in S_2 \\ &etc. \end{aligned} \tag{3.1.3}$$

Recall that the dominant strategy condition for Player 1 is

$$\tilde{U}_1(s_1, s_2, ..., s_n) \geq \tilde{U}_1(t, s_2, ..., s_n) \quad \forall t \in S_1 \text{ and } \forall\, (s_2, .., s_n) \in S_2 \times ... \times S_n$$

which is much more restrictive than (3.1.3) which only requires the inequality to hold for the specific $(s_2, .., s_n)$.

Now we can analyse the above games to look for Nash equilibria.

Examples

- For Rock-Paper-Scissors, there can be no Nash equilibrium in which players use **pure** strategy (see section 3.2 for **mixed strategies**). If Player 1 plays Rock, and Player 2 knew this, Player 2 would play Scissors. But if Player 1 knew Player 2 was playing Scissors, Player 1 would play Rock.
- For Prisoner's Dilemma, the dominant strategy for both players is to accuse the other. If both players have a dominant strategy, this pair is necessarily a Nash equilibrium.
- For Splitting $4, there is a more obvious Nash equilibrium, where each player writes down $2. Note that if either player wrote down $1 while the other wrote down $2, they would receive only $1, while if they write down $3, they receive nothing. So clearly $2 is the best option when the other player writes down $2. However, there are actually two more Nash equilibria: If Player 1 writes down $1 and Player 2 writes down $3 or vice versa. Note that either player will have a worse outcome if they modify their move and the opponent does deviate.
- For Stag hunt, there are two Nash equilibria. If both players hunt stag, they maximize their reward. If both hunt hare, they maximize their reward. Given that the other player is hunting hare, switching to stag would result in a worse outcome.

Determining Nash Equilibria quickly by looking at a table.

Given a game with payoff matrix, for example,

	1	2	3
1	(1,1)	(1,2)	(1,3)
2	(2,1)	(2,2)	(0,0)
3	(3,1)	(0,0)	(0,0)

Player 1 can go through each column, and for each column decide the optimal row, and make a note of this. For example, in the following, each optimal playout for a given column to Player 1 has been starred.

	1	2	3
1	(1,1)	(1,2)	(1^*,3)
2	(2,1)	(2^*,2)	(0,0)
3	(3^*,1)	(0,0)	(0,0)

Player 2 then can do the same thing. For each row, choose the optimal column. (Note that you may star multiple possibilities if they are all equal). As above we get

	1	2	3
1	(1,1)	(1,2)	(1^*,3^*)
2	(2,1)	(2^*,2^*)	(0,0)
3	(3^*,1^*)	(0,0)	(0,0)

We see that there are 3 entries with two stars. Each time we see two stars for an entry, that entry represents a Nash equilibrium.

Consider the matrix game determined by

$$\begin{pmatrix} 3 & 6 & 1 & -4 \\ -3 & 0 & -4 & 7 \\ -1 & 10 & -3 & -1 \\ 5 & -6 & 1 & 5 \end{pmatrix}$$

Each player marking their best strategies (i.e Player 1 marks the maximum in each column with a $^{(1)}$, Player 2 chooses the minimum in each row and marks with a $^{(2)}$) we get

$$\begin{pmatrix} 3 & 6 & 1^{(1)} & -4^{(2)} \\ -3 & 0 & -4^{(2)} & 7^{(1)} \\ -1 & 10^{(1)} & -3^{(2)} & -1 \\ 5^{(1)} & -6^{(2)} & 1^{(1)} & 5 \end{pmatrix}.$$

We observe that there is no Nash equilibrium: This is perhaps unsurprising for zero-sum games. It is possible, however. Consider

$$\begin{pmatrix} 3 & 6 & 10 & 5 \\ -3 & 0 & 7 & -7 \\ -10 & 10 & -8 & -1 \\ 0 & -6 & 1 & 5 \end{pmatrix}.$$

Notice that row 1 and column 1 is a Nash equilibrium: Given player 1 has chosen row 1, column 1 is the minimum, while given that player 2 has chosen column 1, row 1 gives the maximum.

Does this mean that (row 1, column 1) will be the result of the game? Probably, but not necessarily. Each player will be trying to geuss what the other player will play. The only way for Player 1 to do better is to try and correctly guess when and how pPayer 2 is going to guess how Player 1 is going to deviate. But since Player 1 is coming out ahead by not trying to outsmart Player 2, Player 1 has very little incentive to deviate from row 1. So the most likely outcome is the Player 1 will choose row 1. Thus Player 2's best option is column 1.

3.2 Mixed strategies

How would one play a game such as Example 9 in practice? To fully understand the range of strategies, we need to bring in some randomness. This involves the concept of mixed strategies: Each player chooses a strategy that is non-deterministic. This allows for a wider range of strategies, in a continuum of infinitely many possibilities.

For Rock Paper Scissors, instead of choosing a "pure" strategy such as Rock, the player can use a mixed strategy: Roll a six-sided die. If the number shown is 1 or 4, play Rock. If the number is 2 or 5 play Paper. Otherwise, play Scissors. If the die is fair, there is no way for Player 2 to come up with a strategy that will reliably beat Player 1. In fact, each player playing the same $(1/3, 1/3, 1/3)$ random strategy is a Nash equlibrium - even if they both know the other players' strategy, they cannot improve their own odds.

Now to make this more precise we need to talk about how the payoff function works. Instead of a deterministic number we have three possible outcomes, each with a probability. Namely, there are three outcomes. Player 1 wins, Player 2 wins, and tie. To break this down further, notice that there are nine disjoint events that could occur during play, that is, nine samples in our sample space Ω. Define

$$E_{X,Y} = \text{Event that Player 1 plays } X \text{ and Player 2 plays } Y$$

so we have

$$\Omega = \{E_{R,R,}\ E_{R,P,}\ \text{etc.}\}$$

Because the events are disjoint and make up the entire set of possibilities for a round of RPS, we know that

$$\sum_{X\in\{R,P,S\}} \sum_{Y\in\{R,P,S\}} P(E_{X,Y}) = 1.$$

Now among these 9 events there are more general events:

$$\begin{aligned} W_1 &= E_{R,S} \cup E_{S,P} \cup E_{P,R} \\ W_2 &= E_{S,R} \cup E_{R,P} \cup E_{P,S} \\ T &= E_{S,S} \cup E_{R,R} \cup E_{P,P} \end{aligned}$$

and because these three events are disjoint and combine to make the entire set of outcomes, they form a partition, and we have

$$P(W_1) + P(W_2) + P(T) = 1.$$

So now, suppose that Player 1 plays with a mixed strategy: Rock, Paper, Scissor with probability vector (r_1, p_1, s_1) and likewise for Player 2. Because we are assuming the players are making independent random selections, the events, the probabilities can be easily computed:

$$\begin{aligned} P(E_{R,S}) &= r_1 s_2 \\ P(E_{S,S}) &= s_1 s_2 \\ &\text{etc.} \end{aligned}$$

Now the probability that Player 1 wins is precisely

$$P(W_1) = r_1 s_2 + s_1 p_2 + p_1 r_2$$

whereas

$$P(W_2) = r_2 s_1 + s_2 p_1 + p_2 r_1.$$

At this point we observe the following. If we consider the matrix game with matrix

$$A = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$$

we have (from matrix-vector multiplication and the dot product in linear algebra)

$$\begin{aligned} A \begin{pmatrix} r_1 \\ p_1 \\ s_1 \end{pmatrix} \cdot \begin{pmatrix} r_2 \\ p_2 \\ s_2 \end{pmatrix} &= \begin{pmatrix} -p_1 + s_1 \\ r_1 - s_1 \\ -r_1 + p_1 \end{pmatrix} \cdot \begin{pmatrix} r_2 \\ p_2 \\ s_2 \end{pmatrix} \\ &= s_1 r_2 + r_1 p_2 + p_1 s_2 - (p_1 r_2 + s_1 p_2 + r_1 s_2) \\ &= P(W_2) - P(W_1) \end{aligned}$$

Notice this is precisely the expected value that Player 2 will win, namely

$$\mathbb{E}\,[\text{player 2 winnings}] = 1 * P(W_2) + (-1) * P(W_1) + 0 * P(T).$$

This is not a coincidence! In general for a matrix game with matrix A, the expected outcome for Player 2 is always going to be

$$A\vec{p}_1 \cdot \vec{p}_2$$

where $\vec{p}_i$ is the probability vector describing Player i's mixed strategy.

Full exploration of this topic is slightly beyond the scope of this book, but goes back to some of the foundations of modern game theory. A very interesting and noteworthy result of Von Neumann is the following: Every matrix game has a Nash equlibrium if we allow mixed strategies. That is, there is a pair of probability vectors $\vec{p}_1, \vec{p}_2$ such that neither player can improve their expectation, given the other player plays that strategy. A full discussion would take us into to topic of linear programming, also a fascinating and useful area of both pure and applied mathematics.

3.2.1 Mixed strategies in real life

Mixed strategies happen often in real life. Sports is one example. A pitcher in baseball may have a 102 mph fastball, but a batter expecting a fastball can key in on the pitch and get a good swing. So even if a the fastball is the best pitch, the picture will not throw this 100% of the time. In soccer, players shooting penalty kicks need to decide where to kick the ball, left, right or maybe even center. If the goalkeeper anticipates that a player usually kicks right, they can jump right at the instant the ball is kicked, giving themselves a chance to stop the kick.

3.3 Extensive-Form Games

Single play games are easy to set up and analyse, but most real world situations involve a sequence of moves. These are called **extensive-form games**. We know many of these games. Some are deterministic like Chess, while others like Poker are non-deterministic. Chess is of course, extremely difficult to analyse in pure mathematical terms. For blockchain we will see that most games involve some randomness and are extensive-form.

Remark 3.3.1. While we are deliberately vague throughout this text, in this section we are more so. Many of the definition would require quite extensive formal definitions, which we do not provide.

The strategy set for extensive games can be a bit more difficult to notate, but in the most general form can be described a the set of functions from the set of game states to the set of moves. That is, a strategy will be a response to each possible game state. Even though the game is played in ordered,rounds, one can still describe their responses to each possible scenario before arriving there (although typically, you would not reveal this.)

We begin with a very simple, two-round example.

Example 12. ***(Ultimatum Game)*** *Two players are offered a total of \$100, provided they can split the money in the following way. Player 1 proposes a split, and Player 2 either rejects this and both get none, or accepts the split and it is distributed as such. There is no negotation.*

This game has two stages. In Stage 1, Player 1 must decide on the offer. In Stage 2, Player 2 must decide to accept or reject.

Often extensive form games are presented as a game tree. In particular, this is a rooted tree graph with the start of the game as round zero, and edges coming out of this round zero.

In the Ultimatum Game, each possible dollar amount between 0 and 100 represents a move for Player 1, and a corresponding edge coming out of the root node. From each node on round two, there are two option for Player 2, Accept or Reject. So if we assume that Player 1 can only use whole dollar amounts, the game tree will have 202 possible end leaves. For this particular game we can determine the entire outcome of the game by the ending leaf. Some games will require we keep track of the game history a long the way as value can be won or lost.

The first things we notice about this game is that Player 1 has a decided advantage. If Player 2 is an emotionless, rational player who will maximize their winnings from the game, they will certainly accept any positive number. Knowing this basic fact, Player 1 can look at all the possibilities and decide to make a split $(\$99, \$1)$. Since \$1 is preferable to \$0, Player 2 will accept this.

Players 1's strategy set consists of 101 offers, starting from $(\$0, \$100)$ and including $(\$100, \$0)$. Player 2's strategy is more complicated to describe, because Player 2 must have a response to each possible state of the game. There will be 101 such states, so a single strategy for Player 2 will be a function

$$S : \{\text{Player 1's moves}\} \to \{\text{Accept, Reject}\}\,.$$

In practice Player 2's strategy may be quite easy to describe, for example, more prosaically: "Accept all offers of \$1 or more, reject \$0."

Now because of the distinct ordering of the moves of the game, it doesn't really matter what Player 2 declares their strategy will be, in an attempt to persuade Player 1 to offer a better split. Player 2 is forced to make a decision after Player 1 has played, so if acting rationally will probably accept any positive offer.

This is an example of **subgame perfect equilibrium**. A subgame perfect equilibrium is a set of strategies such that these strategies will constitute a Nash equilibrium at every stage of the game. A subgame perfect equilibrium is usually found using **backward induction**, which describes the reasoning Player 1 would have used to make the offer $(\$99, \$1)$.

A strategy is **sequentially rational**, if it for each game state, the strategy maximized the players expected outcomes, given their beliefs about the game. This can be complicated - in game like Chess, the goal is to be as sequentially rational as possible. Doing so is another matter, even for the experts. However in simple examples like the Ultimatum Game, the sequentially rational strategy should be clear. If Player 1 expects that Player 2 will accept any positive offer and will reject a \$0 offer, the only rational strategy is too offer (\$99, \$1).

It's not easy to give a concise definition of backwards induction, short of the rough description "analyse the tree backwards." In the Ultimatum Game, Player 1 can choose their favorite option, namely player 2 accepts a $(\$100, \$0)$ split. The path backwards is clear, Player 2 must accept the offer, that must be first made. Because it isn't certain that Player 2 would accept this offer, Player 1 considers this risky and moves to the next best option, the $(\$99, \$1)$ split. This requires Player 2 accepting the $(\$99, \$1)$ split, which they will do provided they are behaving rationally, in the subgame that starts when Player 1 makes this offer. So Player 1 will make this split.

Here we are discounting psychology or bluffing. Player 2 may indeed reject an offer of less than \$10, out of spite. This isn't rational however, unless the game is repeated and player 2 is attempting to signalling that they will reject inequitable offers, or if Player 2 derives more utility from rejecting insulting offers. We won't discuss bluffing here.

In games we will be discussing we assume players have the capacity to make completely rational decisions. (Unlike, for example, a chess player.) The existence of rational strategies to play chess is a mathematical theorem: A famous result in game theory known as Zermelo's theorem says that "in chess either White can force a win, or Black can force a win, or both sides can force at least a draw". However, nobody can compute the winning strategy, so Chess remains unsolved, at least for the near future.

3.4 Schelling Points

Schelling points, or focal points, are strategies that players will play in absence of communication. Focal points are not acquiescent to a precise mathematical definition. Some maybe more clear, for example, the splitting $4 game, there seems to be a clear focal point - each player writes down $2. This is "fair" and uniquely so, thus intuitively one would imagine the other player would also see this and write down $2.

The original example of a Schelling Point is due Thomas Schelling, who asked the question: If you are supposed to meet someone in New York City, but haven't been able to communicate where or when, where and when would you go to attempt to meet them? In this case, you would attempt to go with what you believed other people would think. In experiments, "Grand Central Station at noon" was the most common answer given. Since this would maximize the probability of meeting the other person, this is probably the best move.

Note that there are similarities in the reasoning for finding Schelling Points and the **Keynesian beauty contest**. In a Keynesian beauty contest, participants are asked to select faces that will be most attractive to judges. It is an interesting fact that people will often collectively have beliefs about others that are wrong. However, because others share the same collective misconceptions the beliefs turn out to create a Schelling Point. For example, players may consider investing in a stock or cryptocurrency. They might have personal opinions about which is stronger on it's own fundamentals, but will often make a investment decisions based on what they think others will find most compelling. Keynes described this (three quarters of century before cryptocurrency) "We have reached the third degree where we devote our intelligence to anticipating what average opinion expects the average opinion to be." [Keynes1936]

Schelling points are chosen because they are special and stand out for some reason. This concept can be quite general and underpin cryptocurrency, or even currency in general. Technically, the notion of Schelling Point requires an absence of communication, so one could argue the concept does not apply. But tomorrow, or the day after, people will continue to use currencies (crypto and otherwise) without holding a collective meeting and deciding to do so. Bitcoin is valuable precisely because people have decided that it is special, and for this reason one could reasonably expect it to maintain a "special" status, at least into the near future. By choosing to transact on the Bitcoin blockchain protocol, you are choosing a "meeting point" where you can meet other users of the same protocol and blockchain. If you vary the protocol or blockchain, you can't expect the same results. This is why the rules of a consensus protocol are crucial. As long we continue to follow the rules of a protocol and prescribe value to coins transferred according to this protocol, we can expect the coin to maintain some status as special.

Chapter 4

Mining games:Preliminaries

Before discussing mining games, we should clear up a few more details about how transactions are structured.

4.1 UTXO

Bitcoin uses something called UTXO, or "unspent transaction output." Bitcoin does not operate based on accounts like a bank. Rather Bitcoin operates based on transactions. Each transaction references an input transaction and creates outputs. The transaction direct the bitcoin in the input to an outspent address, which is a public key or a hash of a public key. When the owner of a corresponding private key would like to spend this output, they create a new transaction, by cryptographically demonstrating they have the private key, and hence are entitled to spend the output, taking this output as the input for a new transaction. This transaction directs the inputs to outputs.

If part of the output is not specified, it can be claimed by the miner who mines the transaction. If the owner of the private key doesn't intend to spend the entire output, they may send the remainder back to themselves, often to another address which they generate. This is one way that the UTXO is different than accounts based accounting - the output of a transaction must be spent when the new transaction is created, even if that means sending the transaction back to yourself. The term "unspent transaction output" refers to the transactions that have not been spent, these have outputs that are spendable.

If you look at a blockchain explorer, perhaps http://blockchain.info, you can click on a random transaction and verify that most transactions will have at least two outputs. The difference between the input and the output will be the fee - this can also be verified by randomly perusing transactions on any blockchain explorer. If you are new to Bitcoin, spending a couple minutes clicking through a blockchain explorer may help you develop a more concrete understanding.

One thing to note is that the transaction structure exists independent of the blockchain. That is, each transaction has an id which does not depend on the block in which it was confirmed. Apart from a given blockchain, there can be many "valid" transactions spending a given output. Here "valid" means only that the transactions are properly signed and are sourced from a valid input. The function of the blockchain is to ensure that there is consensus on which one is the transactions has been accepted, in the case that multiple transaction spend the same output.

Each output can be spent by only one transaction. That transaction may have multiple inputs, or multiple outputs.

One can picture this as directed hypergraph. A directed hypergraph is like a directed graph, but edges can point to, or source from, multiple vertices. Each transaction output is a vertex in this graph. A valid transaction is a directed edge which sends this output to one or more outputs. As this is independent from a blockchain, there can be multiple "valid" transactions which spend the same output, thus the hypergraph has a tree-like structure. (Technically it's not a tree: Trees cannot have loops in the undirected structure, which may become possible when multiple inputs are used in a transaction.) The point is that there can be a large directed graph from transaction outputs to transaction outputs, and the "branch points," where two or more edges source from the same output, represent incompatibilities. A blockchain is valid only when it confirms a connected path of edges that source one from each output.

Apart from the blockchain, there is a simplicity in validating transactions with UTXO; it requires only looking at the preceeding transaction. Each transaction can be traced to previous transactions, and so forth, and this requires no information whatsoever about transactions that occur on other parts of the blockchain. If an "account-based" system were used, the system would have to keep track of all possible inputs and outputs. Instead, the role of nodes and miners is to 1) first check the transaction is valid, and next 2) check that the source transaction output is still unspent.

In creating each block, the miner creates a transaction giving themselves coins for the reward. This comes from "the coinbase." (Note that while here is a company that unfortunately shares this name, "coinbase" refers to the place where coins come from when mined.) Fees also are included in this transaction. Note that because the unspent outputs are not specific transaction to a miner, rather claimed at the time of mining, they won't fall under the independent transaction structure that exists apart from the blockchain. Coinbase transaction are locked for 100 blocks, so that miners cannot immediately spend Bitcoins which they earn.

Nodes typically pool and pass along transactions that have been validated. These transactions form the mempool (memory pool) of valid transactions that could be placed on the next block. Usually these are prioritized by fees.

It is possible for the creator of a transaction to create another transaction with the same output, but with a larger fee. This isn't necessarily considered a double-spend, because the goal might not be to maliciously spend the money twice - rather the goal is to increase the priority of the transaction so that it will be included by the miners. This move is called "replace-by-fee" often abbreviated RBF. This does add an incompatible transaction to the tree described above, but this is OK, only one transaction can be on the accepted blockchain.

Remark 4.1.1. We use the term "valid" as an absolute fact describing whether a transaction is signed correctly, follows protocol and points to a previous valid transaction. Thus validity is a mathematical proposition that is either true or false and cannot change with time. This is completely agnostic of the underlying blockchain which has to accept and confirm the transactions. Valid transaction are valid apart from "the" blockchain or any blockchain, The acceptance of the blockchain as "the blockchain" is a more wrought question as it the question is relative - accepted by whom? In most case the economic majority has clear interest in agreeing on which blockchain is "the" blockchain, but this is not an absolute ground truth.

Now we are ready to set up the game theory underpinning mining.

4.1.1 Common assumptions

We will often, but not always use the following assumptions. More complicated games may require these to be relaxed.

Assumption 1 (near-time price stability) We may assume that the value of a Bitcoin is stable in the near term. For long term we can assume that the price today reflects the discounted future value. This means we don't factor speculation into miners decisions.

Assumption 2 (near-time hashrate stability) We may assume that the total hashrate is stable in the near term.

Assumption 3 (first-seen rule). Miners will always mine on top of the longest chain that they see first, provided there are no other explicit reasons for them not to.

Assumption 4 (instant broadcasting) All nodes and miners are connected enough so that transaction are broadcast nearly instantly.

For the simplest model, consider that there are n mining pools, denoted P_i, and each P_i is capable of producing H_i hashes per second.

At each point in time the state of the game will be the information describing the following

- All known and broadcast competitive chains, including necessarily the longest chain and forks that maybe of comparable length.
- All transactions in the mempool (including their fees), that is, transactions that are valid but may not have been confirmed in the longest chain.
- current hashrate of all of the pools.

Note that *the* mempool is not well-defined. Each mining node has their own mempool that might not match up completely with other nodes. However, for our purposes we can assume there is a single mempool.

Selfish mining, discussed in Chapter 8, will involve exceptions to these assumptions.

First, we assign to each pool a hashrate, h_i via

$$h_i = \frac{H_i}{H}$$

where H is the total hashrate, given by

$$H = \sum_{i=1}^{n} H_i.$$

This means the numbers $h_i \in (0, 1)$ and

$$\sum h_i = 1.$$

Each h_i corresponds to the probability of the event that pool i wins the next block.

The block reward will be given by R. We will denominate all other Bitcoin amounts in terms of this unit. Thus a payment or fee which in 2021 would be expressed as $0.01R$, could also be expressed as 0.0625 BTC. Using R as a base unit allows us to perform computations that are agnostic of the current block reward, which varies over time.

The term **reorg**will refer to an attempt to reorganize the chain, by mining a new chain that starts from a block some blocks behind the current longest chain. This necessarily involves **orphaning** blocks. A block is orphaned if it was once part of the longest chain, and a reorg created a new longest chain that no longer contains this block.

Remark 4.1.2. Suppose that some miners go offline. In the short term, as long as there is no change in difficulty, which updates every two week, there is no greater probability that the miners still online will win blocks. However, there is a greater probability that the remaining miners will win the next block. For example, suppose that suddenly 90% of miners went offline, leaving a single pool of 10%. This pool's odds do not improve of winning a block in the next ten minutes. Until the difficulty changes, this pool expects to win one block every 100 minutes. Now, certainly, they will win the next block, and the block after that, provided that other pools stay offline, but these blocks will remain far apart. The single pool will not see an increase in the frequency at which they win blocks, until the difficulty is adjusted, after the 2016 block period is completed. They may however, see an increase in fees paid, as they may select the transactions with the highest fees to include in the blocks. Blocks will be less frequent so the fees may become more competitive.

Remark 4.1.3. Bitcoin miners and nodes typically run the Bitcoin Core client, downloadable from github. Miners aren't required to, and there are possible other ways to run Bitcoin. For example, there is also libbitcoin, which provides alternative software modules. It's possible that miners or other institutions may be using their own homebrew client. When attacking the problem from a game theory perspective it's easiest to make the assumption that each agent could configure their own software to run a given strategy. This is not completely true; there is some cost to implement one's own version of Bitcoin code. The "average-Joe" node with very little money involved will be unlikely to put forth the effort to modify the code. But a larger party, perhaps a large mining pool, financial institution or a nation-state should have the means to hire someone familiar with the codebase enough to modify the software that they are using to interact with the Bitcoin network. So for this reason we will treat the cost of switching from default strategy to be negligible.

Chapter 5

Monopolizing Pool

As a basic introduction, we consider the case when a single mining pool controls more than 50% of the hashrate. Suppose the pool would like to control the entire blockchain, by mining their own blocks. They would be interested to do this for several reasons: By establishing and then maintaining monopoly power over the mining they can control profit margins - by easing the hashrate down, the difficulty will decrease, while the block reward may or may not decline (unless, of course, such an action causes major dysphoria in the Bitcoin ecosystem; see section 11). Also, the pool may have direct orders from, or be heavily funded by, a larger entity that would like to censor the network's transactions. So to this end, the pool mines only their blocks.

At some point in time, this monopolizing pool (MP, hereafter) decides to start mining only their own blocks, ignoring all other blocks that appear on the network. Aware of this, the other resisting miners (RP) instead start mining their own chain. We define a random variable:

$$\begin{aligned} S_i = &\text{ total blocks found by MP when } i \\ &\text{ blocks have been found by MP and RP} \end{aligned}$$

This will have values in the integers.

It's important to note here what we are keeping track of. We are assuming that both RP and MP are publishing blocks as they find them. The RP is not colluding, and has no motivation to perform a sneak attack on MP, so are best served by instantly publishing all of the blocks. The MP is motivated to demoralize the RP, with the goal of watching all other miners drop out. Publishing blocks is likely to be consistent with this goal.

This is different in cases in which one party is trying to perform a stealth doublespend - mining transactions in secret and then foisting these to the network only after when they have overtaken the main chain and a certain number of blocks have passed. This requires slightly different analysis, see [GP2020].

Part of the simplicity of assuming we know each block that has been produced, is that we don't need to worry about the time variable. If we suspect one pool is mining in secret, the number of blocks they have mined is a random variable itself. By using totals blocked mined as the time unit (roughly, but not consistently 10 minutes) we can obviate this hassle.

5.1 Rough approach: Bernoulli's Law of Large Numbers and Hoeffding's bound

We'll take this opportunity to introduce some useful results from probability theory, and indulge in some very simple proofs.

Theorem 13. *Chebyshev's inequality. Suppose that X is a nonnegative random variable. Then*

$$P\left\{X \geq \varepsilon\right\} \leq \frac{\mathbb{E}\left[X\right]}{\varepsilon}.$$

Proof. Using the conditional expectation formula (2.2.5)

$$\mathbb{E}\left[X\right] = \mathbb{E}\left[X : X \geq \varepsilon\right] P\left\{X \geq \varepsilon\right\} + \mathbb{E}\left[X : X < \varepsilon\right] P\left\{X < \varepsilon\right\}.$$

Certainly the expected value of X, given that $X \geq \varepsilon$, must be at least ε. Because $X \geq 0$, the second term is also nonnegative, and we have

$$\mathbb{E}[X] \geq \varepsilon P(X \geq \varepsilon).$$

Diving by ε gives us the result. $\square$

In particular, if we define

$$\xi_i = 1 \text{ if MP produced the } i\text{th block}$$
$$\xi_i = 0 \text{ if RP produced the } i\text{th block}$$

we have defined a Bernoulli process with $p > .5$ being the hashrate of MP. We then have

$$S_n = \sum_{i=1}^{n} \xi_i.$$

Now by applying (2.2.3) to the independent variables ξ_i,

$$\begin{aligned} Var(S_n) &= \sum_{i=1}^{n} Var\left(\xi_i\right) \\ &= nVar(\xi_i). \end{aligned}$$

We can compute the variation by hand from the definition (2.2.2). We know

$$E(\xi_i) = p$$

so

$$\begin{aligned}(\xi_i - \mathbb{E}[\xi_i])^2 &= (1-p)^2 \text{ with probability } p\\ (\xi_i - \mathbb{E}[\xi_i])^2 &= (0-p)^2 \text{ with probability } 1-p\end{aligned}$$

so

$$\begin{aligned}\mathbb{E}(\xi_i - \mathbb{E}[\xi_i])^2 &= p(1-p)^2 + (1-p)p^2\\ &= p(1-p).\end{aligned}$$

Thus (2.2.3) gives us

$$Var(S_n) = np(1-p).$$

Now by definition

$$\begin{aligned}Var(S_n) &= \mathbb{E}\left[(S_n - \mathbb{E}[S_n])^2\right]\\ &= \mathbb{E}\left[(S_n - pn)^2\right]\end{aligned}$$

that is

$$np(1-p) = \mathbb{E}\left[(S_n - pn)^2\right].$$

We now apply Chebyshev's inequality to the random variable $\left(\frac{S_n}{n} - p\right)^2$:

$$\begin{aligned}P\left\{\left(\frac{S_n}{n} - p\right)^2 \geq \varepsilon\right\} &\leq \frac{\mathbb{E}\left[(\frac{S_n}{n} - p)^2\right]}{\varepsilon} = \frac{\mathbb{E}\left[(S_n - pn)^2\right]}{n^2\varepsilon}\\ &= \frac{p(1-p)}{n\varepsilon}.\end{aligned}$$

The expression given here shows that as n becomes large, the probability that the fraction of blocks obtained by MP differs from p by more than any fixed ε dwindles to 0. This is principle is known as Bernoulli's law of large numbers.

What does this mean in practice? Suppose a mining pool has 51% hashrate. MP has mined over half of the blocks and controls the longest chain if and only if $\frac{S_n}{n} > 0.5$. In particular, this is true whenever

$$\left(\frac{S_n}{n} - p\right)^2 < (0.01)^2.$$

The probability that this condition fails is bounded above by

$$\frac{0.51 * 0.49}{n(0.01)^2} = \frac{2499}{n}.$$

This isn't that great of a bound: If a mining pool has 51% of the hashrate, we have shown that even after 10000 blocks are produced, we can only bound (via this method) the probability that MP has not controlled the blockchain by $\frac{1}{4}$.

This is quite a rough bound, to be fair, we might expect this overestimate the probability by a factor of two because we aren't taking into consideration the fact that $S_n/n > 0.52$ is actually a win for MP.

A more powerful inequality (which we do not prove) is Hoeffding's inequality;

Theorem 14. *(Hoeffding's Inequality) If $\xi_i \in \{0, 1\}$ are independent Bernoulli variables, then*

$$P\left\{\frac{S_n}{n} - p \geq t\right\} \leq e^{-2nt^2}.$$

This statement is much more powerful. To apply it, we switch roles and look at the perspective of the resisting mining pool. In order to maintain control of the chain, we would take $p = 0.49$ and would need

$$\frac{S_n}{n} - p \geq 0.01.$$

So the probability that RP controls the chain is bounded above by

$$e^{-0.0002n}.$$

This may seem to decay slowly at first, but this is exponential decay. After 10000 blocks, this bound becomes

$$e^{-2} \approx 0.135\,34.$$

Observe this is close to half of the bound we obtained previous via Bernoulli's law of large numbers. This continues to decay exponentially fast from there: After 20000 blocks the probability is

$$e^{-4} \approx 0.01831\,6.$$

Of course, the control of a chain by MP would be much more drastic if they had a larger percentage: 51% means eventually they will control the chain, but not necessarily today or tomorrow. The assurance that this chain will be ahead (from this bound) does not even approach 99% after ten weeks.

However, if MP controlled 60% of the hashrate the computation changes drastically. By Bernoulli's law of large numbers, we get

$$P\left\{\left(\frac{S_n}{n} - p\right)^2 \geq 0.01\right\} \leq \frac{E\left[(S_n - pn)^2\right]}{\varepsilon} = \frac{24}{n}.$$

After 144 blocks (roughly on day) the probability that the number of blocks is between 50 and 52% is 5/6. Using Hoeffding's inequality

$$P(\frac{S_n}{n} - p \geq t) \leq e^{-.02n}$$

In comparison, after 1000 blocks, the Hoeffding bound assures us that the probability that MP has failed to control the chain is bounded by 2.06×10^{-9}.

$$e^{-.02*1000} \approx 2.06 \times 10^{-9}.$$

5.2 Binomial distribution and de Moivre-Laplace theorem.

To be more precise, we may apply the theory in section 2.3.1.

From the perspective of MP, they are ahead provided that

$$S_n > \frac{n}{2}.$$

Thus

$$\begin{aligned} P\left(S_n > \frac{n}{2}\right) &= 1 - P\left(S_n \leq \frac{n}{2}\right) \\ &= 1 - \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{i} p^i (1-p)^{n-i}. \end{aligned}$$

This is possible to compute by hand with a computer. Taking $n = 1000$ and $p = .51$

$$1 - \sum_{i=0}^{500} \binom{1000}{i} (0.51)^i (0.49)^{1000-i} \approx 0.726\,10$$

Taking $n = 10000$ and $p = .51$.

$$1 - \sum_{i=0}^{5000} \binom{1000}{i} (0.51)^i (0.49)^{1000-i} \approx 0.9767 \tag{5.2.1}$$

This can be done easily using a number of methods. For example, the python package `scipy.stats` has a function `binom.cdf`, which performed the above computation in a split-second. If you are able to say "Hello World" with python, you should be able open up a session and

```
from scipy.stats import binom
binom.cdf(5000, 10000, 0.51)
```

and the result will be the probability that a mining pool with 51% hashrate has created more than 5000 or less of the first 10000 blocks.

For practical purposes, this is probably sufficient. A basic installation of python generates a floating point truncation error at about $n = 10^9$ which is more blocks than will be produced in any of our lifetimes.

If you find yourself stuck in 1960s, you may use the de Moivre-Laplace theorem as an approximation. Recall that the random variable

$$\tilde{S}_n = \frac{(S_n - np)}{\sqrt{np(1-p)}}$$

converges to a random variable with standard normal distribution. The normal distribution function $\Phi(x)$ has been computed for decades by hand and is available in many reference books. For $p = 0.51$, $n = 10000$ we are interested in

$$\begin{aligned} P(S_n) &\leq 5000 \\ &= P\left((S_n - np) \leq 5000 - 5100\right) \\ &= P\left(\frac{(S_n - np)}{\sqrt{np(1-p)}}\right) \leq \frac{-100}{\sqrt{10000(0.49)(0.51)}} \approx -2.000\,4 \\ &= P\left(\tilde{S}_n \leq -2.000\,4\right) \\ &= \Phi\left(-2.000\,4\right) \approx 0.02\,272\,9 \end{aligned}$$

Observe that

$$1 - 0.02\,272\,9 = 0.977\,27$$

is rather close to the value obtained in (5.2.1).

The de Moivre-Laplace theorem is useful for following both p and n as they both change, or in solving one based on knowledge of the other.

Example 15. *Find the smallest value of p that will ensure with probability at least 90% that the mining pool is ahead by 10 blocks after the first 10000 blocks have been mined.*

Solution: We want to find p such that

$$P\left(S_n \geq 5005\right) = 0.9.$$

We compute

$$\begin{aligned} P\left(S_n \geq 5005\right) &= P\left(\tilde{S}_n \geq \frac{5005 - 10000 * p}{\sqrt{1000p(1-p)}}\right) \\ &= 1 - P\left(\tilde{S}_n < \frac{5005 - 10000 * p}{\sqrt{1000p(1-p)}}\right) \\ &= 1 - \Phi\left(\frac{5005 - 10000 * p}{\sqrt{10000p(1-p)}}.\right) \end{aligned}$$

At this point we solve

$$1 - \Phi\left(u\right) = 0.9$$

or

$$\Phi\left(u\right) = 0.1$$

which can be done by consulting any "Table of the Standard Normal Cumulative Distribution Function" and determine that

$$u \approx -1.29.$$

Then we solve

$$\frac{5005 - 10000p}{\sqrt{10000p(1-p)}} = -1.29$$

and get

$$p = 0.506\,95.$$

Chapter 6

Basic double spend game

Suppose that Alice controls a transaction output, and double spends this. She creates a valid transaction and sends this to Bob, and then creates another valid transaction sending this to Charlie. While these transactions might both be valid, they can not go on the same blockchain. Miners are left to decide, perhaps arbitrarily, which transaction to include in the next block. This need not be random, they can choose transaction that has the higher fee, or choose the transaction which came to their attention first.

Let's consider the simplest possible scenario, Bob operates mining pool P_1 while Charlie operates mining pool P_2. In this case each will certainly try to confirm their own transaction, while the remaining pools may go either way. In a few minutes, one of the transactions will be confirmed. Without loss of generality, suppose that this is Bob's transaction. Thus at the latest block, Bob is in possession of the transaction output sent by Alice. What should Charlie do?

Most miners, by default will simply continue to append to blockchain. Charlie, who controls h_2 of the total hashrate, needs to decide if it's worthwhile to try and fork the blockchain at the previous block, in hopes to overtake and surpass the current chain. The default rule follwed by miners is that they continue to mine on the first-seen longest chain. So not only does Charlie need to catch up; Charlie must create a longer blockchain in order for miners to begin building on Charlie's block.

We can analyze Charlie's probablility of accomplishing this, and decide whether it is worth doing so. Even though it appears somewhat simple, the question is slightly involved. It's an extensive form game in that in many situations the game will not conclude after the first round. Charlie can choose from a variety of strategies. The simplest is to forget the transaction. More risky strategies are to continue to mine even when many blocks behind.

We assume that the payment to Charlie that Charlie wants to claim is λR. In our first computation we will ignore not only Bob's motivations, but the accumulated block rewards of different mining pools and focus only on the difference in length. (The situation where Bob decides to fight back is more complicated.

This will be initially be discussed at the end of this chapter in section 6.2, and will be revisited later in the context of Nash Bargaining. Note that if Bob's hashrate is small it is fair to ignore this case, at least for the time being.) The other miners will be thought of as non-playable-characters, essentially following the first-seen longest chain rule at all times, even if that means orphaning a block they have mined in the past. Thus the instant that Charlie's chain becomes longer, Charlie wins. For this game, it will be convenient to think in terms of stages, each new stage occurring when a new block is produced on either chain. In the simplest form, this state information is just the number describing the difference between the lengths of chains. While we may be tempted to think we need to keep track of who mined the block first, this only matters when the two chains are equal, and because the opposing chains would not have mined a chain after Charlie took the lead, we can assume that equal lengths means that Charlie is not first-seen. If Charlie's was first seen, the game would be over and Charlie would have won. The game state should also include information about how many blocks Charlie has produced on his own chain.

We define the state space as follows

$$\left\{(i,j) \mid \left(\begin{array}{c} i \in \{-1,0,1,2,...\} \text{ denotes how far behind Charlie's chain is} \\ j \in \{0,1,2,...\} \text{ denotes how many blocks Charlie has mined} \end{array} \right)\right\}. \tag{6.0.1}$$

For example, the game starts with state $(1,0)$, denoting that Charlie is one block behind, and has produced zero blocks since the game started. Any state of the form $(-1,j)$ is terminal: This means Charlie has overcome the main chain and taken the lead.

Denoting now $p = h_2$ and hereafter, the probability that the state moves from $(1,0)$ to $(0,1)$ is p while the probability that the state moves from $(1,0)$ to $(2,0)$ is $1-p$. (See figure)

This can generalized: the probability that the first state value decreases by 1 will always be p. The probability that it increases by 1 is $1-p$. The second state vector increases by 1 every time the first state vector decreases by 1 and stays the same otherwise.

Charlie has an array of strategies available to him. For each possible state of the game, Charlie can either continue to pursue mining his own blockchain, or give up and go back to mining the main chain. One very basic set of strategies available to Charlie are "mine only when not behind by more than k blocks." This strategy only considers the first state variable and might not necessarily allow Charlie the optimal strategy: The second variable is not purely a "sunk-cost", because it is recoverable if (and only if) Charlie proceeds to win.

The probability space we work on will be the Bernoulli space $\mathcal{B}$, which consists of all sequences chosen from

$$\{\text{Charlie finds next hash, Someone else finds next hash.}\}$$

Recall the remarks in section 2.3.2, in particular the left-hand/right-hand game comparison. There are two universes, with equivalent probability spaces $\mathcal{B}$: one in which Charlie employs a fixed but divergent strategy, and one in which

Charlie simply mines the longest chain. Because these probability space are equivalent we may ignore the fact that the particular outcomes in $\mathcal{B}$ are different in the two different universes (recall section 2.3.2,) and define a random variable as follows

$$\begin{aligned} f_{stategic} &= \text{How many block rewards Charlie wins} \\ &\quad \text{using the strategy when the game ends} \\ f_{null} &= \text{How many block rewards Charlie would} \\ &\quad \text{have won when the game ends using null strategy} \end{aligned}$$

And then define

$$f = f_{stategic} - f_{null}.$$

Again the "would have won" is a slight technical abuse: It doesn't quite make sense because the game would never have been played. But due to the remarks in section 2.3.2 we may ignore this technicality.

Clearly, if

$$E(f) > 0$$

then Charlie has chosen a better strategy than the null strategy and he should play this. Thus we seek to maximize, if possible, the value $E(f)$ over all possible strategies. Unfortunately, the it's not so simple to develop a formula for each possible strategy. Short of maximizing over all strategies, we are happy to find at least one strategy that yields a positive expectation. We start with a very simple strategy "mine only when not behind by more than k blocks.".

Remark 6.0.1. We are presenting the analysis from Charlie's perspective, which is omniscient. It's possible in general that a double spend attempt may involve a secret fork, see section 6.1. Suppose you want perform a double spend by sending an output to yourself and also to an exchange. You keep the output to youself secret, and mine this output onto a chain in secret, hoping to spring your chain onto the network once you have been credited with by the exchange and are able to withdraw elsewhere, or take possession of whatever you purchased. The analysis from the viewpoint of the exchange (how many blocks should pass before we credit the transaction in order to minimize risk?) is slightly more complicated as it involves the time variable as well. This is covered in works by Cyril Grunspan and Ricardo Pérez-Marco [GP2018, Theorem 5.2][GP2020].

Remark 6.0.2. Some of the presentation covers topics similar to that covered in the work [GP2018][GP2020]. However, we are assuming that attackers are using a loss-cutting strategy, and will attempt to compute how much they can be expected to lose, rather than simply computing the probability that they win.

6.0.1 Simplest Strategy: Charlie elects to mine his own chain when less than k blocks behind.

This game starts at state $(1, 0)$, meaning that Charlie is one block behind, and has yet to mine any blocks on his forked chain. Charlie commits to a strategy of mining his own chain until the moment that his chains fall k blocks behind, at which point, Charlie will switch back to the main chain. This may or may not be optimal, but this is involved enough to compute, so we begin here.

First of all, we would like to know the probability that Charlie will win, given the current state. Let's define notation

$$w_i = \text{Probability that Charlie will win, given that Charlie is } i \text{ blocks behind.}$$

It is important to notice that in this straightforward strategy model, w_i depends on the first state variable and not the second, because all subsequent hashes produced are independent of the previous ones.

Recursion formulas.

The analysis depends on the conditional probability formula (2.2.5). Note that by letting W denote the random variable that is 1 if Charlie wins and 0 otherwise we have

$$w_i = \mathbb{E}\left[W \mid \text{Charlies is } i \text{ blocks behind}\right].$$

If Charlie is i blocks behind, there are two events that could happen immediately next, and we know these probabilities. So we write

$$\begin{aligned}
&\mathbb{E}\left[W \mid \text{Charlie is } i \text{ blocks behind}\right] \\
&= \mathbb{E}\left[W \mid \begin{matrix}\text{Charlie is } i \text{ blocks behind} \\ \text{and wins the next block}\end{matrix}\right] P\left(\text{Charlie wins the next block}\right) \\
&+ \mathbb{E}\left[W \mid \begin{matrix}\text{Charlie is } i \text{ blocks behind} \\ \text{and loses the next block}\end{matrix}\right] P\left(\text{Charlie loses the next block}\right)
\end{aligned}$$

by (2.2.5). But

$$\mathbb{E}\left[W \mid \begin{matrix}\text{Charlie is } i \text{ blocks behind} \\ \text{and loses the next block}\end{matrix}\right] = \mathbb{E}\left[W \mid \text{Charlie is } i+1 \text{ blocks behind}\right]$$

so we can write in simpler terms

$$w_i = pw_{i-1} + (1-p)w_{i+1}. \tag{6.0.2}$$

This is true for any i in between -1 and k, and it becomes most useful if we start working from the end points. We know that

$$w_{-1} = 1$$

which simply states that if Charlie is -1 blocks behind he has pulled ahead, and all miners now mine his chain, he has won. At the other end,

$$w_k = 0 \tag{6.0.3}$$

stating the Charlie gives up the fight and concedes defeat when he is k blocks behind.

There are multiple ways to proceed in solving for the values w_i. We will offer two. The first method is somewhat straightforward and gives nice-looking formulas for smaller values of k. The second method is involved, using difference equations and is somewhat more high-tech. The formulas are not as pretty but this method will lend itself to more complicated computations in later sections, and will be easy to program using linear algebra.

A method for computing w_i

Begin by noting that

$$\begin{aligned} w_{k-1} &= pw_{k-2} + (1-p)w_k \\ &= pw_{k-2} \end{aligned}$$

from (6.0.3). Proceed with the goal to continue to define w_i terms of w_{i-1}. So write

$$w_i = q_i w_{i-1} \tag{6.0.4}$$

and we have determined that

$$q_{k-1} = p.$$

With this in mind, we may write, using (6.0.2) (6.0.4)

$$\begin{aligned} w_{i-1} &= pw_{i-2} + (1-p)w_i \\ &= pw_{i-2} + (1-p)q_i w_{i-1} \end{aligned}$$

or

$$w_{i-1}\left(1-(1-p)q_i\right) = pw_{i-2}$$

which can be solved

$$w_{i-1} = \frac{pw_{i-2}}{(1-(1-p)q_i)}$$

that is

$$q_{i-1} = \frac{p}{(1-(1-p)q_i)}.$$

We can iterate this, until we get to

$$\begin{aligned} w_0 &= q_0 w_{-1} \\ &= q_0. \end{aligned}$$

Where

$$\begin{aligned} q_0 &= \frac{p}{(1-(1-p)q_1)} \\ q_1 &= \frac{p}{(1-(1-p)q_2)} \\ &\dots \\ q_{k-1} &= p \\ q_k &= 0. \end{aligned} \tag{6.0.5}$$

For example, when $k = 2$, we have

$$\begin{aligned} q_1 &= p \\ q_0 &= \frac{p}{(1-(1-p)p)}. \end{aligned}$$

Hence

$$\begin{aligned} w_0 &= \frac{p}{(1-(1-p)p)} \\ w_1 &= \frac{p^2}{(1-(1-p)p)}. \end{aligned}$$

Exercise 16. *Let $m = -(1-p)p$. Use (6.0.5) to show that there is a sequence of polynomials $\rho_i(m)$ such that for a given k,we have*

$$q_{k-i} = p\frac{\rho_{i-2}(m)}{\rho_{i-1}(m)}$$

$$\begin{aligned} w_0 &= p\frac{\rho_{k-2}(m)}{\rho_{k-1}(m)} \\ w_1 &= p^2\frac{\rho_{k-3}(m)}{\rho_{k-1}(m)} \\ &\dots \\ w_{k-1} &= p^k\frac{\rho_{-1}(m)}{\rho_{k-1}(m)} \end{aligned}$$

where

$$\begin{aligned} \rho_{-1}(m) &= 1 \\ \rho_0(m) &= 1 \end{aligned}$$

and the polynomials satisfy the recursion relation for $i \geq 1$

$$\rho_i(m) = \rho_{i-1}(m) + m\rho_{i-2}(m).$$

Another method: difference equations While the above by-hand method will yield a formula for w_i in any given situation, there is another observation that offers us a streamlined computation. Consider the equation,

$$w(x) = pw(x-1) + (1-p)w\,(x+1) \tag{6.0.6}$$

thinking here of w as a function of the variable x on the interval $[-1, k]$. The equation appears to be somewhat like a second order finite difference equation. Drawing from intuition from second order homogenous ODEs (cf. [Lebl2014]) we take a guess of the solution of the form

$$w(x) = e^{\lambda x}$$

in which case (6.0.6) becomes

$$e^{\lambda x} = pe^{\lambda x - \lambda} + (1-p)e^{\lambda x + \lambda}.$$

Dividing by $e^{\lambda x}$

$$1 = pe^{-\lambda} + (1-p)e^{\lambda}$$

which can be solved

$$e^{\lambda} = \frac{p}{(1-p)}$$

hence

$$w(x) = \left(\frac{p}{1-p}\right)^x$$

is a solution to (6.0.6). Also note that if w is constant (6.0.6) will be solved. So we define two solutions

$$\begin{aligned} w_1(x) &= \left(\frac{p}{1-p}\right)^x \\ w_2(x) &= 1 \end{aligned}$$

which are independent solutions (two functions are independent if one function is not simply a multiple of the other.) Noting that the equation (6.0.6) can be written as a linear homogeneous equation

$$w(x) - pw(x-1) - (1-p)w\,(x+1) = 0$$

we may easily check two solutions added together will also be a solution. By general theory for homogeneous equations, as in the case of linear second order ODEs (cf. [Lebl2014, Section 2.1]) we can write the general solution as

$$w(x) = c_1\left(\frac{p}{(1-p)}\right)^x + c_2.$$

By prescribing boundary values

$$\begin{aligned} w(-1) &= 1 \\ w(k) &= 0 \end{aligned}$$

we may solve for the coefficients

$$c_1 \left(\frac{p}{1-p}\right)^{-1} + c_2 = 1 \tag{6.0.7}$$

$$c_1 \left(\frac{p}{1-p}\right)^{k} + c_2 = 0 \tag{6.0.8}$$

and get

$$c_1 = \frac{p}{1-p} \frac{1}{\left(1 - \left(\frac{p}{1-p}\right)^{k+1}\right)} \tag{6.0.9}$$

$$\begin{aligned} c_2 &= -\left(\frac{p}{1-p}\right)^{k+1} \frac{1}{\left(1 - \left(\frac{p}{1-p}\right)^{k+1}\right)} \\ &= -\left(\frac{p}{1-p}\right)^{k} c_1. \end{aligned} \tag{6.0.10}$$

Hence

$$w(x) = \frac{\left(\frac{p}{1-p}\right)^{x+1} - \left(\frac{p}{1-p}\right)^{k+1}}{1 - \left(\frac{p}{1-p}\right)^{k+1}}. \tag{6.0.11}$$

For simplicity of computation in the future, we will let

$$m = \frac{p}{1-p}$$

and note that the coefficients are determined by the solution to the following linear system, which is no more than using a matrix to write the equations (6.0.7)(6.0.8) as

$$\begin{pmatrix} m^{-1} & 1 \\ m^k & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{6.0.12}$$

Remark 6.0.3. The above expression (6.0.11) is reminiscent of the formula for computing sums of geometric series. This is not coincidence. Most of the results in this section and the next could be derived using geometric series.

When $k = 2$ we have

$$w(0) = \frac{\left(\frac{p}{1-p}\right)^{1} - \left(\frac{p}{1-p}\right)^{3}}{\left[1 - \left(\frac{p}{1-p}\right)^{3}\right]}$$

$$w(1) = \frac{\left(\frac{p}{1-p}\right)^{2} - \left(\frac{p}{1-p}\right)^{3}}{\left[1 - \left(\frac{p}{1-p}\right)^{3}\right]}.$$

While this method allows us to jump immediately to the formula (6.0.11) it has the disadvantage that having a more cumbersome expression.

Computing expected values

Now we are ready to attempt to compute the expected value to Charlie of the function f, where we define

$f =$ the proceeds of mining his fork strategy $-$ the proceeds from the null strategy.

We are interested in

$$\mathbb{E}\left[f \mid (1,0)\right]$$

that is, the expected benefit of mining the reorg strategy when starting from 1 block behind.

From state $(1,0)$ there are two possible next states: $(2,0)$ and $(0,1)$ so we write, again using (2.2.5):

$$\mathbb{E}\left[f \mid (1,0)\right] = p\mathbb{E}\left[f \mid (0,1)\right] + (1-p)\mathbb{E}\left[f \mid (2,0)\right].$$

Introducing a less cumbersome notation :

$$e(x,s) = \mathbb{E}\left[f \mid (x,s)\right]$$

this relation becomes

$$e(1,0) = pe(0,1) + (1-p)e(2,0).$$

Now, at this point we could continue to write all of these difference equations and recursively solve in either of the methods given above. For brevity, we only present the method using the difference equations. Unlike in the previous example, when there was only one state variable, in this case, we have two. Our method is similar to as before. We begin by observing the relation:

$$e(x,s) = pe(x-1,s+1) + (1-p)e(x+1,s)$$

Claim 17.

$$e(x,s+1) = e(x,s) + w(x) - 1. \tag{6.0.13}$$

Proof. Given we are at state (x,s) and following a given strategy that only depends on x, the probability space of possible paths starting at (x,s) and going until the game ends is identical to the probability space of possible paths starting at $(x,s+1)$. For each path that ends in success, the reward to Charlie is same in both case, because all of the block rewards that would have been won along the way are granted when he takes the longest chain. But if the path ends in a loss for Charlie, he loses all block rewards he would have won had he pursued the null strategy. He loses one more block reward if the path started from state $(x,s+1)$ than if the same path had started from path (x,s). The difference will be the probability of a loss, which is $1-w(x)$. □

So now the equation becomes

$$\begin{aligned} e(x,s) &= p\left[e(x-1,s) + w(x-1) - 1\right] + (1-p)e(x+1,s) \\ &= pe(x-1,s) + (1-p)e(x+1,s) + p\left(w(x-1) - 1\right). \end{aligned}$$

Note this now only involves a single value in the second state variable. So if we are interested in computing $e(x,0)$ we may define

$$e(x) = e(x,0)$$

which must satisfy

$$\begin{aligned} e(x) - pe(x-1) - (1-p)e(x+1) &= p\left(w(x-1) - 1\right) && (6.0.14) \\ &= p\left(c_1\left(\frac{p}{1-p}\right)^{x-1} + c_2 - 1\right) && (6.0.15) \end{aligned}$$

for the c_1 and c_2 obtained above in (6.0.9). This is similar to the finite difference equation (6.0.6) in x, but with extra terms on the right-hand-side, making this a non-homogenous equation. As for second order linear non-homogenous ODEs, we will try to find a particular solution, and then add this to the solutions w_1 and w_2 obtained previously to obtain the general solution.

First we consider the non-homogeneous equation

$$y(x) - py(x-1) - (1-p)y(x+1) = p(c_2 - 1). \qquad (6.0.16)$$

We try a very basic guess (this is similar to the method of undetermined coefficients from ODE theory)

$$y_1 = \beta x$$

for some constant β. Plugging in and solving for β we get

$$\beta = \frac{p(c_2-1)}{2p-1}.$$

Thus

$$y_1 = p\frac{1-c_2}{1-2p}x$$

solves (6.0.16). Next we solve

$$y(x) - py(x-1) - (1-p)y(x+1) = pc_1\left(\frac{p}{1-p}\right)^{x-1} \qquad (6.0.17)$$

For this we try a solution of the form

$$y_2 = \alpha x\left(\frac{p}{1-p}\right)^{x}.$$

Plugging y_2 into (6.0.17) and doing some straightforward computations leads to

$$\alpha = \frac{c_1(1-p)}{1-2p} \qquad (6.0.18)$$

We conclude that

$$y_2 = \frac{(1-p)}{(1-2p)} c_1 x \left(\frac{p}{1-p}\right)^x$$

Is a solution to (6.0.17).

Thus we arrive at the general solution to (6.0.14)

$$y(x) = \tilde{c}_1 \left(\frac{p}{(1-p)}\right)^x + \tilde{c}_2 + p\frac{1-c_2}{(1-2p)}x + \frac{(1-p)}{(1-2p)} c_1 x \left(\frac{p}{1-p}\right)^x.$$

To be clear, c_1 and c_2 are the coefficients that were previously determined, while $\tilde{c}_1$ and $\tilde{c}_2$ are coefficients that need to be solved when finding a solution to a boundary value problem. We can then solve for the boundary conditions:

$$\begin{aligned} e(-1) &= \lambda \\ e(k) &= 0. \end{aligned}$$

(Recall that when Charlie takes the lead, he wins the blocks he would have won, plus the extra λ times the block reward. If Charlie gives up mining without having won a single block, he is no better or worse off than if he mined the null strategy.)

Solving the boundary conditions determines $\tilde{c}_1$ and $\tilde{c}_2$ by

$$\tilde{c}_1 \left(\frac{p}{(1-p)}\right)^{-1} + \tilde{c}_2 - \frac{p(1-c_2)}{(1-2p)} - \frac{(1-p)}{(1-2p)} c_1 \left(\frac{p}{1-p}\right)^{-1} = \lambda \tag{6.0.19}$$

$$\tilde{c}_1 \left(\frac{p}{(1-p)}\right)^{k} + \tilde{c}_2 + \frac{p(1-c_2)}{(1-2p)}k + \frac{(1-p)}{(1-2p)} c_1 k \left(\frac{p}{1-p}\right)^{k} = 0 \tag{6.0.20}$$

It is convenient to write this as a 2×2 system of equations

$$\begin{aligned} \begin{pmatrix} m^{-1} & 1 \\ m^k & 1 \end{pmatrix} \begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \end{pmatrix} &= \begin{pmatrix} \lambda + mr(1-c_2) + rc_1 m^{-1} \\ -mr(1-c_2)k - rc_1 k m^k \end{pmatrix} \\ &= \lambda \begin{pmatrix} 1 \\ 0 \end{pmatrix} + r \begin{pmatrix} m(1-c_2) + c_1 m^{-1} \\ -m(1-c_2)k - c_1 k m^k \end{pmatrix} \\ &= \lambda \begin{pmatrix} 1 \\ 0 \end{pmatrix} + r \begin{pmatrix} m^{-1} & m \\ -km^k & -km \end{pmatrix} \begin{pmatrix} c_1 \\ 1-c_2 \end{pmatrix}. \end{aligned}$$

once again using less cumbersome notation

$$\begin{aligned} m &= \frac{p}{1-p} \\ r &= \frac{1-p}{1-2p}. \end{aligned}$$

with

$$\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} m^{-1} & 1 \\ m^k & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{6.0.21}$$

Here

$$\begin{pmatrix} m^{-1} & 1 \\ m^k & 1 \end{pmatrix}^{-1} = \frac{1}{m^{-1} - m^k} \begin{pmatrix} 1 & -1 \\ -m^k & m^{-1} \end{pmatrix}$$

is the inverse matrix from linear algebra. Now

$$\begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \end{pmatrix} = \begin{pmatrix} m^{-1} & 1 \\ m^k & 1 \end{pmatrix}^{-1} \begin{pmatrix} \lambda + mr(1-c_2) + rc_1 m^{-1} \\ -mr(1-c_2)k - rc_1 k m^k \end{pmatrix}. \tag{6.0.22}$$

At this point we may plug the constants into the expression to obtain the function.

$$e(x,0) = \tilde{c}_1 m^x + \tilde{c}_2 + mr\,(1-c_2)\,x + rc_1 x m^x. \tag{6.0.23}$$

We choose not to do expand this out: the expression is messy and not particularly enlightening.

Examples: p=0.1, k small

In this example, a miner with 10% of the hashrate considers a strategy to mine a reorg chain if 2 or less blocks behind, and give up if 3 blocks behind.

Taking $p = 0.1$, $k = 3$ gives us

$$m = \frac{1}{9}$$
$$r = \frac{9}{8}$$

$$c_1 = \frac{1}{9}\frac{1}{1-\frac{1}{9}^4} = \frac{729}{6560} \tag{6.0.24}$$
$$c_2 = -\frac{1}{9}^4 \frac{1}{1-\frac{1}{9}^4}. = -\frac{1}{6560}$$

$$\begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \end{pmatrix} = \frac{\frac{1}{9}}{1-\frac{1}{9}^4} \begin{pmatrix} 1 & -1 \\ -\left(\frac{1}{9}\right)^3 & \frac{1}{9}^{-1} \end{pmatrix} \begin{pmatrix} \lambda + \frac{1}{9} * \frac{9}{8} * (1 + \frac{1}{6560}) + \frac{9}{8} * \frac{729}{6560} * \left(\frac{1}{9}\right)^{-1} \\ -\frac{1}{9} * \frac{9}{8} * (1 + \frac{1}{6560}) * 3 - \frac{9}{8} * \frac{729}{6560} * 3 * \left(\frac{1}{9}\right)^3 \end{pmatrix}$$
$$= \begin{pmatrix} \frac{729}{6560}\lambda + \frac{1554\,957}{8606\,720} \\ -\frac{1}{6560}\lambda - \frac{3234\,573}{8606\,720} \end{pmatrix}$$

And

$$e(x) = \left(\frac{729}{6560}\lambda + \frac{1554\,957}{8606\,720}\right)\left(\frac{1}{9}\right)^x - \frac{1}{6560}\lambda$$
$$- \frac{3234\,573}{8606\,720} + \frac{1}{8}\left(1 + \frac{1}{6560}\right)x + \frac{9}{8} * \frac{729}{6560} x \left(\frac{1}{9}\right)^x.$$

We can plug in the values for x :

$$\begin{aligned} e(-1) &= \lambda \\ e(0) &= \frac{91}{820}\lambda - \frac{6561}{33\,620} \\ e(1) &= \frac{1}{82}\lambda - \frac{729}{3362} \\ e(2) &= \frac{1}{820}\lambda - \frac{405}{3362} \\ e(3) &= 0. \end{aligned}$$

We conclude that in order to mine with these odds, a miner starting with 1 block deficit is only motivated to attempt a reorg if

$$e(1) = \frac{1}{82}\lambda - \frac{729}{3362} > 0 \tag{6.0.25}$$

that is

$$\lambda > \frac{729}{41} \approx 17.\,78$$

and for a 2 block deficit

$$\lambda > \frac{4050}{41} \approx 98.\,78.$$

In other words, for a pool with 10% of hashrate, in order for it to be prudent to attempt to perform a reorg with a "mine to three blocks behind" strategy, when starting from two blocks behind, the reward would need to be roughly 100 times the block reward.

On the other hand, if a mining pool mines a transaction, only to discover that all the other pools have discovered another block seconds earlier, in order to push forward, this pool is wise to do so if

$$\lambda > \frac{6561}{3731} \approx 1.\,758\,5.$$

We can ask the same question, but for $k = 2$:
Similar calculations yield

$$\begin{aligned} e(-1) &= \lambda \\ e(0) &= \frac{10}{91}\lambda - \frac{729}{8281} \\ e(1) &= \frac{1}{91}\lambda - \frac{810}{8281} \\ e(2) &= 0. \end{aligned}$$

So if the pool adopts a strategy to mine until two blocks behind, starting from 1 block behind, this makes sense if

$$\lambda > \frac{810}{91} \approx 8.\,901.$$

The following table shows the "break-even" multiple of the block-reward required to mine the reorg strategy when starting from 1 block behind:

	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 5$
$p = .1$	8.9011	17.780	27.546	37.507	47.501
$p = .2$	3.8095	7.5294	11.976	16.769	21.696
$p = .3$	2.0675	3.9425	6.3118	9.0544	12.048
$p = .4$	1.1842	2.0769	3.1822	4.4973	6.0114
$p = .49$	0.70766	1.0824	1.4714	1.8749	2.2929

You may observe that the chart is monotone: break-even numbers increase as k becomes larger. However, this only tells part of the picture. It's also interesting to look at the expected value for a fixed λ. For the following we take a fixed $\lambda = 10$ and compute the expected advantage or disadvantage in mining the reorg chain.

	$k = 2$	$k = 3$	$k = 4$	$k = 5$
$p = .1$	0.01208	-0.0949	-0.2163	-0.3395
$p = .2$	0.29478	0.14533	-0.12169	-0.42150
$p = .3$	0.90370	0.93995	0.63324	0.16885
$p = .4$	1.8560	2.4378	2.4557	2.1514

This is more instructive as to how Charlie would behave in practice. Given the extra reward λ, Charlie plugs in p to the formula for several values of k. The value of k that maximizes the expected surplus will be the strategy. It appears (we make no attempt to prove this here) that function exhibits concave behavior with respect to k, in which case there is a unique maximizer k which can be found by increasing k until the function begins to decrease.

There is an improvement in this strategy that will be difficult to optimize precisely, but can be described. Suppose Charlie is sent a transaction worth 10 times the block reward, but fails to process it in time, and must peform a reorg starting from one block behind. The above analysis says that Charlie is wise to pursue a strategy, starting from time 0, to mine until two blocks behind, and then give up. However, there is (somewhat rare) possibility that Charlie will have a string of alternating blocks with the rest of chain. If this continues for long enough, Charlie's reward for winning as opposed to giving up will be not only the reward λ but $\lambda + s$ where s is the number of blocks that Charlie has won. So there may be an opportunity to refine the strategy as the game continues to get a slightly better strategy. So the estimates proved here are lower bounds that are slightly but strictly below the optimal strategy. For p closer to $\frac{1}{2}$ and k larger, this becomes more pronounced. The computation of this optimized strategy is left as an exercise for the very ambitious reader.

In the most general sense, Charlie's strategy is a choice of stopping points, given in both state variables. To be precise we define

$$\text{Stopping Set} = \{\text{All states } (k, j) \mid \text{Charlie will not mine the reorg chain at this state.}\}$$

Notice this includes not only the boundary states that might be reached during an attempt, but also includes those that will never be reached in a reorg attempt.

Again, while this is difficult to compute precisely, we can build a hierarchy of stopping points: If $(2,0)$ is a not stopping state, then $(2,1)$ is not a stopping state: the reward for winning is larger in the latter point. In general

$$(k, j+1) \in \text{Stopping Set} \implies (k, j) \in \text{Stopping Set}$$

Now it would be a silly strategy for Charlie to say, mine his own chain when the state is $(4, j)$, but then switch to mining the main chain when the state is $(3, j+1)$ so we can also observe that

$$(k, j+1) \in \text{Stopping Set} \implies (k+1, j) \in \text{Stopping Set}.$$

Case $k = 7$.

Six blocks is commonly thought of as "safe" in the sense that transactions are "confirmed" once six blocks have been mined starting with the block mining the transaction. In this section we use the calculus developed above to demonstrate this principle. Again, part of the issue that complicates this is that we may not know if someone is attacking the network in secret or not. So first, we proceed from the easier and more naive perspective of the potential attacker who will be making the decision to attack or not from already 6 blocks behind.

First note the following, if we look only at the chance of winning when a mining pool commits to mining forever, even if they are very far behind, their odds are given by the function $w(x)$ when by taking the limit

$$w_\infty(x) = \lim_{k\to\infty} \frac{\left(\frac{p}{1-p}\right)^{x+1} - \left(\frac{p}{1-p}\right)^{k+1}}{\left[1 - \left(\frac{p}{1-p}\right)^{k+1}\right]} = \left(\frac{p}{1-p}\right)^{x+1}. \tag{6.0.26}$$

In particular, if the a pool with hashrate p starts k blocks behind and commits to trying catching up and winning, never cutting losses, their chances of catching up are

$$\left(\frac{p}{1-p}\right)^{k+1}.$$

Remark 6.0.4. Often the probability of simply catching up ([GP2018, Lemma 3.1]) is computed, this is $\left(\frac{p}{1-p}\right)^{k}$. We require *catching up and winning.*

For example if $p = 0.3$ the chance of overcoming a six block deficit is

$$\left(\frac{0.3}{0.7}\right)^7 \approx 0.0026556$$

For $p = .15$ the chances are much lower:

$$\left(\frac{0.15}{0.85}\right)^7 \approx 5.329\,7 \times 10^{-6}$$

but better than 1 in a million!

We also see that, as $p \to \frac{1}{2}$ the odds of overcoming a large deficit don't decrease quite as rapidly, for example, if we let $p = 0.4$ and $k = 20$

$$\left(\frac{.4}{.6}\right)^{21} \approx 2.0049 \times 10^{-4}.$$

This may make sense when compared to familiar sports events: A basketball team that should otherwise have a .400 winning probability may surge 21 games above .500. Rare, but not quite a black swan.

In fact, we observe that (6.0.26) tells us the following, if the probability is exactly $p = 1/2$, the probability of winning is 1 regardless of starting position. This is equivalent the statement that an unbiased random walk on the set of integers will eventually hit every integer, excepting certain perversely chosen sequences that happen with probability 0.

Now these probabilities don't factor in the cost. It's likely that such an attempt would bankrupt a mining pool. So even a 1 in a 10,000 chance will likely be not worth the cost, when many block rewards will have been spent in pursuing this. This is why we need to use the more involved formulas to determine the expected gain or loss.

The follow table is the λ required to "break-even" when starting from 6 blocks behind:

	$k = 7$	$k = 8$	$k = 9$	$k = 10$	$k = 11$
$p = .1$	672596	1210680	1796068	2391839	2989396
$p = .2$	7270.2	11638.7	16632	21918	27320
$p = .3$	477.27	673.71	900.35	1148.87	1411.7
$p = .4$	69.892	87.617	107.59	129.64	153.58
$p = .45$	30.986	36.502	42.559	49.157	56.293
$p = .49$	16.423	18.138	19.910	21.740	23.628

Exercise 18. *(involved) Compute formula for $e(1, 0)$ for $k = 2$ and p using convergent geometric sequences. Hint 1: Partition the probability space of Bernoulli processes into events described by which round the game ends. The game ends with probability 1 so this is possible. Find the pattern and write it as as a geometric series. Hint 2: Compute and compare the Taylor series for the functions $\frac{1}{1-x}$ and $\left(\frac{1}{1-x}\right)^2$ and use this to rewrite the expression you see.*

6.1 Stealth 6 block attacks.

The attack described immediately above is unlikely, because it starts from a position significantly behind. A more feasible attack would be to immediately begin mining a secret chain, keeping this from the network until the appropriate time.

For example, some exchanges will credit an account bitcoin after 6 blocks have passed confirming the transaction. This is a common practice, but as pointed out in [Rosenfeld2014] and [GP2020] this doesn't accurately account for the probability of success of the attackers, without considering the time as well. An exploit that one may try is the following. Send a large transaction to an exchange, say 100 bitcoins. Once the credit has been applied to your account on the exchange, immediately trade the currency (say for ETH) and then withdraw ETH (or just withdraw bitcoin is allowed.) Then attempt to annul the original large transaction. If the exchange has policies in place that pause larger withdrawals, there may be ways around this. For example, setting up other ringer accounts with sky-high sell orders in a very thinly traded currency (like say CANN - CannabisCoin). One may purchase the thinly traded currency with Bitcoin ("fat-finger" orders are common), and then withdraw the Bitcoin from the ringer account, perhaps days or months later.

The best way to pull this off is to attempt to mine a competing transaction, and have the exchange transaction ready-to-go. To begin, you must control a large output. You compute but don't publish a transaction directing this output to the exchange. Meanwhile, you have a secret transaction that directs this same output to an address you control. You mine this transaction is secret, until you have a block. Once you have obtained a block that includes the self-directed transaction, you immediately publish the exchange-directed transaction, but not the block you have mined. At this point, you have a one-block lead against the main chain. Your goal is to secretly maintain that lead until the main chain has been extended by 6 blocks. After the main chain has mined 6 blocks, the exchange considers your deposit good. If your secret chain is 7 blocks longer, you can go forwards with whatever shenanigans you had in mind. Of course, you may always abort. If you haven't mined any more secret blocks, you can forget the whole operation and consider it just a waste of an hour's worth of mining power (and possible transaction fees encountered along the way.)

The analysis of this is done by computing the probability of each of the different states that the game can be in when the main chain adds 6 blocks. Then for each of these states, we can apply the computation we have already done. There are 6 events to measure. One event is that the attacker has added 6 or more blocks (so has 7 total) by the time that the main chain gets to 6. This outcome is clear, the attacker wins by immediately publishing their 7 block extension. If the attacker has produced 5 more secret blocks, this sets off a tie situation as in the computations above, we must compute $w(0)$ to determine the probability that the attacker will win, and $e(0, 5)$ for the expected net winning. In turn, we make computation for each of the other possible states.

As before will assume that the loss-cutting strategy is determined by a constant k. This is probably a simplification of the optimal strategy: In practice an attacker who has built 10 secret blocks and is 8 blocks behind may be more likely to proceed than say, the attacker who is simply 8 blocks behind having only produced a single secret block.

To keep things simple, we will assume that $k = 6$.

The computation will involve what's called the **negative binomial distribution** formula (cf. [GP2018, Section 5]).

6.1.1 Negative binomial distribution

The negative binomial distribution models the number of successes in a sequence of Bernoulli trials before a specified number of failures occurs. In attacker's perspective, "success" is the attacker obtaining a block. This happens with probability p. A "failure" is the main chain adding a block. This happens with probability $1-p$. We are interested in knowing the probabilities of the events that there are $0, 1, 2, 3, 4, 5$, or more than 6 successes, at the moment that the 6th failures occurs.

In general the negative binomial distribution describes the probability that in the first $x+r-1$ trials, there have been $r-1$ failures and x successes, and then the $x+r$-th trial results in a failure. The distribution is given by

$$\phi(x; r, p) = \binom{x+r-1}{r-1} p^x (1-p)^r$$

here

$$\phi(x; r, p) = P(\text{number of success} = x \text{ when the } r\text{th failure happens}).$$

Remark 6.1.1. Different versions of this formula exist and sometimes the roles of success and failure have been swapped. Also note that

$$\binom{x+r-1}{r-1} = \binom{x+r-1}{x}$$

which may make some formulas found online or in the literature appear differently.

Thus the probability that the attacker has mined x blocks when the main chain reaches its sixth is given by (note here that we are assuming that 1 block has already been credited to the attacker)

$$\phi(x; 6, p) = \binom{x-1+5}{5} p^{x-1} (1-p)^6.$$

As before, we let f be the net winnings for the attack. This will involve the reward λ, which is the amount of the double spend that has been recovered in a successful attack. We want to compute the probability of attaining this and account for the expected number of mined blocks that will be lost in a failed attempt. Since we are assuming $k = 6$ we can then use a straightforward strategy.

Define events

$$Q_i = \text{attacker has mined } i \text{ blocks when the main chain reaches 6}$$
$$Q_{7+} = \text{attacker has mined 7 or more blocks when the main chain reaches 6}$$

$$\begin{aligned}\mathbb{E}[f] = &\sum_{i=1}^{6} \mathbb{E}\left[f \mid Q_i\right] P(Q_i) \\ &+ \lambda P(Q_{7+}).\end{aligned}$$

The values $P(Q_i)$ are given precisely (notice that we are starting at 1 block)

$$\begin{aligned} P(Q_i) &= \binom{i+4}{5} p^{i-1}(1-p)^6 \\ &= \phi(i-1;6,p) \\ P(Q_{7+}) &= 1 - \sum_{i=1}^{6} P(Q_i). \end{aligned}$$

Now we compute $\mathbb{E}\left[f \mid Q_i\right]$. Note that Q_i means, in the convention set up in (6.0.1) that we are at state $(6-i, i)$. Fortunately, we have begun to compute the function $e(x, s)$ already. Recall we know (6.0.23) and (6.0.13)

$$e(x,0) = \tilde{c}_1 m^x + \tilde{c}_2 + mr\,(1-c_2)\,x + rc_1 x m^x \tag{6.1.1}$$

$$e(x,s+1) = e(x,s) + w(x) - 1. \tag{6.1.2}$$

Applying iteratively,

$$e(x,i) = e(x,0) + i(w(x) - 1).$$

So

$$\begin{aligned} \mathbb{E}\left[f \mid Q_i\right] &= e(6-i, i) \\ &= e(6-i, 0) + i(w(6-i) - 1). \end{aligned}$$

We can combine this all into

$$\begin{aligned} \mathbb{E}[f] = &\sum_{i=1}^{6} \left[e(6-i,0) + i(w(6-i) - 1)\right] \phi(i-1;6,p) \\ &+ \lambda \left(1 - \sum_{i=1}^{6} \phi(i-1;6,p)\right). \end{aligned} \tag{6.1.3}$$

Here coefficients in (6.1.1) are determined by (6.0.21,6.0.22).

Computations

We are interested to know if a double spend attack described above pays off: Namely the attacker has hash-power p and gives up at 6 blocks behind, in search of a double spend value of λ block rewards. We will compute the break even value of λ. The computation proceeds as follows.

1. Compute c_1, c_2, and $\tilde{c}_1, \tilde{c}_2$, by (6.0.21,6.0.22).
2. Compute values $w(0), ..., w(6)$ and $e(0,0), e(1,0), ..., e(6,0)$ using (6.1.1).
3. Plug into (6.1.1) to get a an expression of the form

$$\mathbb{E}[f] = a(p,6)\lambda - b(p,6).$$

4. The break even value is
$$\lambda = \frac{b(p,6)}{a(p,6)}.$$

Note that negative binomials are easily computed with the `scipy.stats.nbinom` python package.

The following table describe the multiple of the block reward a double spend that would have steal for it to be wise to pursue.

hashrate	break-even
0.025	5849594.63919
0.05	119708.096744
0.075	13694.1085119
0.1	3163.39696518
0.125	1072.16913626
0.15	462.049924291
0.175	234.325080226
0.2	133.404850221
0.225	82.6362287125
0.25	54.4772052334
0.275	37.5948737631
0.3	26.8071499673
0.325	19.5368113771
0.35	14.4129053512
0.375	10.6666920629
0.4	7.85014186382
0.425	5.69532558725
0.45	4.03873239636
0.475	2.7768062571

Observe that with 30% of the hashrate, a pool would be incentivized to try this strategy if the reward is 27 times the block reward. In 2021, this is 170 bitcoins would make this profitable.

6.2 Extensive game: When pools duel over the double spend

If Bob and Charlie have comparable hashrates, it would make sense that if Charlie was able to claim the payment by winning a reorg race, Bob would be similarly motivated. If this is true, it diminishes Charlies expectation: The game might not actually be over after he pulls ahead. While the main chain will mine his blocks, and Charlie now has a decisive advantage, he still has to worry about the possibility that Bob will come from behind and re-position the original transaction on the main chain. We can model this. The game tree is fairly simple; at each stage either the challenger wins, and the game continues, or the challenger loses and the game is over for good.

Let

$$\begin{aligned} q &= P(\text{Charlies wins a given round as the challenger}) \\ \bar{q} &= P(\text{Bob wins a given round as the challenger}). \end{aligned}$$

Round 1 is the initial challenge by Charlie. Let f denote the net expectation of winnings to Charlie as before. We have

$$\mathbb{E}\left[f \mid \text{Round 1}\right] = q\mathbb{E}\left[f \mid \text{Charlie Wins Round 1}\right] + (1-q)\,\mathbb{E}\left[f \mid \text{Charlie Loses Round 1}\right]$$

and

$$\mathbb{E}\left[f \mid \text{Charlie Wins Round 1}\right] = \left\{ \begin{array}{c} (1-\tilde{q})\mathbb{E}\left[f \mid \text{Bob Loses Round 2}\right] \\ +\bar{q}\mathbb{E}\left[f \mid \text{Bob Wins Round 2}\right] \end{array} \right\}$$

Combining, noting that Bob loses Round 2 means the game is over and Charlie wins λ

$$\mathbb{E}\left[f \mid \text{Round 1}\right] = \left\{ \begin{array}{c} q\tilde{q}\mathbb{E}\left[f \mid \text{Round 3}\right] + q(1-\bar{q})\lambda \\ +\,(1-q)\,\mathbb{E}\left[f \mid \text{Charlie Loses Round 1}\right]. \end{array} \right\}$$

Letting

$$\begin{aligned} e(1) &= \mathbb{E}\left[f \mid \text{Round 1}\right] \\ l(1) &= \mathbb{E}\left[f \mid \text{Charlie Loses Round 1}\right] \end{aligned}$$

we get

$$e(1) - q\bar{q}e(3) = q(1-\bar{q})\lambda + (1-q)l(1). \tag{6.2.1}$$

This could be written as a first order difference equation. Rather the solving an equation, we can find another relation between $e(1)$ and $e(3)$. This will allow us to solve directly for $e(1)$.

The initial value $l(1)$ can be computed. Observe that the expectation for a single round of the game such as (6.0.25) is of the form

$$A\lambda - B.$$

Using the condition expectation formula

$$A\lambda - B = \left\{ \begin{array}{c} P(\text{wins the round})\lambda \\ -P(\text{loses the round})\mathbb{E}[\text{blocks lost} \;\mid\; \text{round is lost}] \end{array} \right\}.$$

we conclude

$$l(1) = \mathbb{E}[\text{blocks lost} \;\mid\; \text{round is lost}] = \frac{B}{P(\text{loses the round})}. \tag{6.2.2}$$

Now we would like to compute the difference δ in

$$e(3) = e\,(1) - \delta.$$

Because the game trees going forward from stage 1 and stage 3 are isomorphic, the difference δ will determined by the expected number of blocks that were expended in getting from stage 1 to stage 3.

6.2.1 How many blocks expected to finish a winning round?

Let

$$\begin{aligned} X &= \text{number of blocks expended in Round 1, if Charlie wins} \\ X &= 0, \text{ if Charlie loses} \end{aligned}$$

Then

$$b(x,s) = \mathbb{E}[X, \text{ given state } (x,s)].$$

Clearly

$$\begin{aligned} b(-1,s) &= s \\ b(k,s) &= 0 \end{aligned}$$

and

$$b(x,s) = pb(x-1,s+1) + (1-p)b(x+1,s).$$

As before (recall claim 17)

$$b(x-1,s+1) = b(x-1,s) + w(x-1).$$

So we have an equation

$$b(x,s) - pb(x-1,s) - (1-p)b(x+1,s) = pw(x-1).$$

with general solution

$$y(x,0) = \tilde{c}_1 m^x + \tilde{c}_2 - \frac{pc_2}{1-2p}x + \frac{1-p}{1-2p}c_1 x m^x$$

where c_1, c_2 are solved in finding w as in (6.0.21). Solving for

$$\begin{aligned} b(-1,0) &= 0 \\ b(k,0) &= 0 \end{aligned}$$

gives us the equation

$$\begin{aligned} \begin{pmatrix} m^{-1} & 1 \\ m^k & 1 \end{pmatrix} \begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \end{pmatrix} &= \begin{pmatrix} -c_2 mr + c_1 r m^{-1} \\ c_2 mrk - c_1 krm^k \end{pmatrix} \\ &= r \begin{pmatrix} m^{-1} & -m \\ -km^k & mk \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}. \end{aligned}$$

So

$$\begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \end{pmatrix} = r \begin{pmatrix} m^{-1} & 1 \\ m^k & 1 \end{pmatrix}^{-1} \begin{pmatrix} m^{-1} & -m \\ -km^k & mk \end{pmatrix} \begin{pmatrix} m^{-1} & 1 \\ m^k & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Now

$$\mathbb{E}[X, \text{ given state } (1,0)] = \tilde{c}_1 m + \tilde{c}_2 + rm\left(c_1 - c_2\right).$$

We then compute

$$\begin{aligned}\mathbb{E}[X,\text{ given state }(1,0)] = w(1)&\mathbb{E}[X,\text{ given state }(1,0)\text{ and Charlie wins}]\\ &+(1-w(1))\mathbb{E}[X,\text{ given state }(1,0)\text{ and Charlie wins}]\end{aligned}$$

Since the variable is set up to vanish in the second case, we have

$$\mathbb{E}[X,\text{ given state }(1,0)\text{ and Charlie wins}] = \frac{\tilde{c}_1 m + \tilde{c}_2 + rm\,(c_1-c_2)}{w(1)}.$$

This will be the first term in the difference between $e(3)$ and $e(1)$.

How many blocks expected to be lost in losing effort?

This is a similar question. Given that Bob challenges Charlie's victory, and Bob wins, how many blocks can we expect Charlie to have lost?

Let

X = number of blocks expended by the non-challenging pool in Round 2, if Bob wins.
$X = 0$ if Bob loses.

We will divide X by Charlie's share of the not-Bob pool to get the number we would like.

Define

$$b(x,s) = \mathbb{E}[X,\text{ given state }(x,s)].$$

Here the state is determined from Bob's perspective: x is how many blocks Bob is behind, s is how many blocks he has mined. Clearly

$$\begin{aligned}b(-1,s) &= s\\ b(k,s) &= 0\end{aligned}$$

and

$$b(x,s) = \bar{p}b(x-1,s+1) + (1-\bar{p})b(x+1,s)$$

where $\bar{p}$ is Bob' hashrate. Aping the computation above, we get

$$\mathbb{E}[X,\text{ given state }(1,0)\text{ and Bob wins}] = \frac{\tilde{c}_{\bar{1}} m + \tilde{c}_{\bar{2}} + rm\,(\bar{c}_1-\bar{c}_2)}{\bar{w}(1)}.$$

where

$$\begin{pmatrix}\tilde{c}_{\bar{1}}\\ \tilde{c}_{\bar{2}}\end{pmatrix} = \bar{r}\begin{pmatrix}\bar{m}^{-1} & 1\\ \bar{m}^{k} & 1\end{pmatrix}^{-1}\begin{pmatrix}\bar{m}^{-1} & -\bar{m}\\ -k\bar{m}^{k} & \bar{m}k\end{pmatrix}\begin{pmatrix}\bar{m}^{-1} & 1\\ \bar{m}^{k} & 1\end{pmatrix}^{-1}\begin{pmatrix}1\\ 0\end{pmatrix}$$

for all barred quantities being determined by $\bar{p}$ instead of p.

Finishing the computation

We now have

$$\begin{aligned} e(3) - e(1) &= \frac{\tilde{c}_1 m + \tilde{c}_2 + rm\left(c_1 - c_2\right)}{w(1)} + \frac{p}{1-\bar{p}} \frac{\tilde{c}_{\bar{1}} m + \tilde{c}_{\bar{2}} + rm\left(\bar{c}_1 - \bar{c}_2\right)}{\bar{w}(1)} \\ &= -\delta. \end{aligned} \tag{6.2.3}$$

noting that Charlie accounts for

$$\frac{p}{1-\bar{p}}$$

of the not-Bob pool.

Returning to (6.2.1)

$$e(1) - q\bar{q}e(3) = q(1-\bar{q})\lambda + (1-q)l(1). \tag{6.2.4}$$

we plug in (6.2.3):

$$e(1) - q\bar{q}\left(e(1) - \delta\right)) = q(1-\bar{q})\lambda + (1-q)l(1).$$

This can be solved

$$e(1) = \frac{q(1-\bar{q})\lambda + (1-q)l(1) - q\bar{q}\delta}{(1-q\bar{q})}.$$

Here $l(1)$ is determined by 6.2.2.

We will return to this and offer more computations in our discussion of Nash Bargaining.

Chapter 7

Censorship attacks

Censorship attacks can happens in a number of ways. We are going to explore what portion of the hashrate is necessary to perform an attack and how that influences the fee that must be paid in order to over come the incentives against including the transaction on the blockchain.

The least effective way for a pool or collection of pools to censor is to simply to decline to process a transaction. Even if the pool has a majority of the hashrate, anything less than 100% hashrate will allow the transaction on the chain. This is why Bitcoin is often described as **censorship resistant.** Effective censorship requires a more aggressive strategy. To effectively censor, the censoring parties should commit to orphaning any blocks. If the censoring parties have more than 51% they should be able to accomplish this in the long run. If the censoring pool has less than 51% of the hashrate, they should not expect to be able to overwhelm rest of the hashrate on a consistent basis, at least by brute force. Even if they accomplish this feat once, or twice, the transaction will stay in the mempool and should eventually be confirmed.

The most effective approach for the censoring party is to announce their intention to attempt to orphan any blocks containing undesirable transactions, and then allow incentives to influence other rational miners. At this point, presuming the censors can be taken seriously with this commitment, the information will be considered by other miners. Before getting into the details, we can explain this with a quick example. Suppose that a pool with 25% of the hashrate announces that they will attempt to orphan any transactions which spend a certain output. Assume this threat is received as credible by other miners on the network. Now suppose a transaction appears in the mempool. The non-censoring miners have two options: Leave it alone, or attempt to mine it. If the transaction fee is small or non-existent, the miners have little or nothing to gain by including the transaction when attempting to construct a block. On the other hand, if they do include the transaction in their next block, there will be a risk: There is a least a $\frac{1}{16}$ chance that the censoring pool will immediately write two blocks on top of the latest compliant block, before anyone appends to the offending block. So the expected value for mining such a block is at least

1/16 of a block reward less. In order to compensate for this the transaction should offer a fee of at least 1/16 of a block reward, or expect that miners are either ignorant of, or resistant to, the censorial pool's threats to orphan the block.

More precise modeling with more details makes the computations more complex. We will consider three sets of mining pools. The censorial (C) mining pool will have hashrate p and is not interested in minimizing costs. They will follow a fixed predetermined strategy, with primary objective of keeping offending transactions off of the blockchain. There may also be a non-compliant (NC) pool, which has hashrate q, which is not interested in censorship for ideological or technical reasons and so will always follow the longest chain rule, and will mine any transactions. Finally there will be a rational (R) set of pools. These may have different sizes $\varepsilon_1 + ... + \varepsilon_n = r$ and these are considered economic free agents: They will join the compliant pool when more profitable and proceed with non-compliant mining when profitable.

The NC pool will be assumed to not use any strategy: simply mine the longest chain. The C pool will declare their strategy at the beginning, which will be simple, and other players in the game will take this to be credible. For example, the C pool can declare that they will attempt to orphan all non-compliant blocks that are no more than k blocks deep, but will not pursue reorgs beyond a certain depth. It is then for the remaining rational pools to determine their respective strategies. The setup is complicated to we start with a simple warm up.

Of course, if the C pool owns 51%, they control the network. They can orphan any block, eventually, as is their prerogative. If the C pool has less than 50% it makes sense to use a loss-cutting strategy. There's no sense pursuing an increasingly unlikely attempt to orphan a specific transaction indefinitely.

7.1 Worked example: two rational pools

Suppose we have only two rational pools of sizes $\varepsilon_1, \varepsilon_2$ so that

$$p + q + \varepsilon_1 + \varepsilon_2 = 1.$$

We assume that p is less than 50%. Suppose that C declare that their strategy is to attempt to orphan any blocks that are less than k blocks deep. If k blocks behind they will drop the fight and look for more recent forking point, or mine the chain tip with compliant transactions. With this declared strategy we will attempt to begin to build a "payoff table" for a few basic possible strategies for the two rational pools. For simplicity, assume that the distasteful transactions are happening somewhat infrequently, so that there we will not expect to see a series of consecutive blocks containing offending transactions that may demand a more sophisticated censoring strategy.

The game starts when the discountenanced output holder attempts to send a transaction. The unspent output (fee) will be given by λ in units of the block reward. We will assume that NC has positive hashrate. This assumption means that eventually the NC will win k consecutive blocks and the transaction will be accepted. In practice, depending on the strategies of the rational pools and the size of NC, this terminal event may occur within an hour, or it may take centuries. However it's a convenient mathematical assumption as we can then model the game as a game that terminates in finite time with probability 1.

Remark 7.1.1. As a side note, we point out that there is a huge mathematical difference between the words "finite" and "bounded." Unfortunately these words are often used interchangeably. For example, saying the supply of a currency is always finite is usually a true statement, whereas saying the supply is "bounded" (as specified in Bitcoin's whitepaper) is a completely different statement. "Bounded" requires a number that provides the upper bound. Similarly, saying that a game terminates in finite time with probability 1 can be true, even if the expectation of the time it take is infinite. The payoff of the St. Petersburg Paradox game is finite, but is not bounded.

From a rational pool's perspective, there are two questions to be asked:

1. What is the expected number of block rewards that a rational pool would waste in fighting the censors?

2. What is the probability that they receive the extra reward from the fee?

Multiplying the answer to the latter by λ and subtracting the former serves as a useful metric in comparing different strategies.

We can think of the game as occuring in a series of rounds. Each round begins when the offending transaction is mined, and terminates one of two ways. The round may terminate with the compliant chain pulling a full block ahead, in which case the game continues. The NC pool attempts to mine the offending transaction on the chain tip, instead of making a come-from-behind attempt. The next round of the game starts when the the transaction is again mined on the chain tip. The round may also terminate with the NC chain pulling ahead k blocks. In this case, the round terminates and so does the game. So to compute the expected outcome for a given pool, we divide the analysis into these two cases.

In each of the scenarios determined by the pools' strategies, there will be a function $w(x)$, which will have to be computed, that determines the probability that the censorial pool will win the round. When the transaction is mined and the round begins, the probability that the game will continue beyond that round is $w(1)$. The probability that the game terminates is $1 - w(1)$. We can compute the expected value that a pool will have won in the latter case. In the former case we will need to set up some recursion relationships to complete the computation.

In particular, after fixing a strategy regime, and a pool Pool i, define a random variable $W_i \in \{0, 1\}$ via

$$W_i = 1 \text{ if Pool } i \text{ claims the fee } \lambda \text{ when the game ends}$$
$$W_i = 0 \text{ otherwise}$$

and a random variable $L_i \in \mathbb{Z}_+$

$$L_i = \text{ blocks mined by Pool } i \text{ that have been orphaned when the game ends.}$$

Then we define the random variable (dropping subscripts when clear)

$$f = \lambda W - L.$$

The goal then is to compute

$$\mathbb{E}[f] = \lambda \mathbb{E}[W] - \mathbb{E}[L].$$

Now

$$\mathbb{E}[f] = \left\{ \begin{array}{c} (1 - w(1))\, \mathbb{E}[f \mid \text{C loses round 1}] \\ +w(1)\mathbb{E}[f \mid \text{C wins round 1}] \end{array} \right\} \tag{7.1.1}$$

and then

$$\begin{aligned} \mathbb{E}[f \mid \text{C wins round 1}] = (1 - w(1))\, \mathbb{E}[f \mid \text{C loses round 2}] \\ + w(1)\mathbb{E}[f \mid \text{C wins round 2}]. \end{aligned} \tag{7.1.2}$$

Now if we define a quantity

$$\tau = \mathbb{E}[\text{Blocks lost by Pool } i \text{ in any given round} \mid \text{C wins that same round}],$$

we have

$$\mathbb{E}[f \mid \text{C wins round 1}] = \mathbb{E}[f] - \tau. \tag{7.1.3}$$

We also use notation

$$\sigma = \mathbb{E}[f \mid \text{C loses round 1}]$$

and combine (7.1.2) with (7.1.3) to get

$$\mathbb{E}[f] = (1 - w(1))\, \sigma + w(1)\, (\mathbb{E}[f] - \tau)$$

which reduces to

$$\mathbb{E}[f] = \sigma - \tau \frac{w(1)}{1 - w(1)}. \tag{7.1.4}$$

In the following sections, our goal will be to compute the values σ, τ and the function $w(x)$.

7.1.1 Consideration for this choice of function.

One difficulty in modeling a game like this is coming up with an appropriate quantity to measure. We've made the decision to essentially model the game as a one-shot affair. The miners in question will mine until the game is over and then return to mine as they had been before the game started. In reality, this may or may not be optimal. In particular, we are not considering situations in which new offending transaction will pop up while a previous transaction is being fought over.

The goal of the computations will be to demonstrate that, given a fee, and a hashing power of the censorial pool, the rational pools can compare different strategies. The results will be meaningful with the assumption that the game will be ending in a reasonably short period of time. (This is not an unreasonable assumption.)

For this reason, the values that appear in forthcoming payoff tables involve only the expected gain or loss from what would be mined in that time period, and are not considering how much time is taken to complete the game.

7.1.2 There are many strategies

The number of strategic responses can be quite numerous. We focus on some of the easier to describe and implement.

1. (Non-compliant) First, there is the non-compliant at all times strategy. The pool simply ignores the censors and includes the transaction, and always mines the longest chain using the first-seen rule.

2. (Mildly compliant) A second strategy would be to attempt to build on the censoring fork when the difference is 0 blocks but not when 1 block deep. In particular, the pool would never mine the offending transaction always leaving it be while in the mempool, and will break ties for the censors.

3. (More compliant) A third strategy would be to not only exclude the transaction, but attempt to build on the censoring fork when the difference is 0 or 1 blocks, and go back to longest chain if two blocks deep.

4. (Neutral/selective). The pool my decide to abstain from mining when the difference is 1 block deep, not wanting to risk the event this block would be orphaned. If the energy cost is significant, these sorts of strategies may come into play.

There are many more variations. Computing outcomes with each of these would give us a 4×4 payoff table with 16 outcomes. Even with symmetries, this would already lead to a somewhat lengthy computation, so we restrict to the first 2 strategies. This will result in a 2×2 payoff table with essentially 4 computations, due to symmetries.

We will attempt to fill in the following table.

Pool 2 strategy / Pool 1 strategy	Non-compliant	Mildly Compliant
Non-compliant	?	?
Mildly Compliant	?	?

7.1.3 Case 1: (Non-compliant, Non-compliant)

We consider the game from the perspective of Pool 1, the payoff computation to Pool 2 computation would be identical. From Pool 1's perspective, there is no difference between Pool 2 and NC, as both employ the same behavior.

Our goal is to compute σ and τ from the perspective of Pool 1, given this strategy regime. The first state variable, i, will refer to how far behind the compliant chain is: Then $i = -1$ denotes the fact that the compliant chain has won the round, while $i = 0$ means the NC chain has been equalized by the compliant chain, so that the "first-seen" rule favors the NC chain. The numbers $\{1, 2, .., k\}$ denote that the C chain is behind by this number of blocks. At block k C will concede the transaction.

The second state variable, s, will refer to the number of block rewards Pool 1 has contributed to the NC chain.

The third variable, which is determined and doesn't change after the first block is mined, will be the amount that Pool 1 has won by mining the offending transaction, taking values in $\{0, \lambda\}$. This doesn't need to be dragged through computations along the game, but is necessary information when the game ends in favor of the NC chain.

While the game begins when the objectionable transaction is posted, the round begins (as with all subsequent rounds) when the transaction is mined and the longest chain becomes non-compliant. The block will either have been mined by Pool 1, Pool 2 or NC. With probability $\varepsilon_1/(1-p)$ the round begins in state $(1,1)$ and with $1-\varepsilon_1/(1-p)$ probability the game begins in state $(1,0)$. We split into these cases to begin computing the expected value of L_1 :

$$\mathbb{E}[L_1] = \frac{\varepsilon_1}{1-p}\mathbb{E}[L_1 \mid (1,1)] + \left(1 - \frac{\varepsilon_1}{1-p}\right)\mathbb{E}[L_1 \mid (1,0)]. \tag{7.1.5}$$

Using conditional expectations,

$$\mathbb{E}[L_1 \mid (x,s)] = \left\{ \begin{array}{c} p\mathbb{E}[L_1 \mid (x-1,s)] \\ +(q+\varepsilon_2)\,\mathbb{E}[L_1 \mid (x+1,s)] \\ +\varepsilon_1\mathbb{E}[L_1 \mid (x+1,s+1)] \end{array} \right\}. \tag{7.1.6}$$

This incorporates the three possibilities; the next block is won by the C, NC/Pool 2, or Pool 1 respectively. We may use the fact that can be concluded from argument similar to the one leading to Claim 17:

$$\mathbb{E}[L_1 \mid (x,s+1) = \mathbb{E}[L_1 \mid (x,s)] + w(x). \tag{7.1.7}$$

Letting

$$e_1(x) = \mathbb{E}[L_1 \mid (x,0)],$$

we have from (7.1.6) and (7.1.7) with $s = 0$

$$\begin{aligned} e_1(x) &= pe_1(x-1) + (q+\varepsilon_2)e_1(x+1) + \varepsilon_1\left(e_1\left(x+1\right) + w(x+1)\right) \\ &= pe_1(x-1) + (1-p)e_1(x+1) + \varepsilon_1 w(x+1). \end{aligned}$$

Because for the entirety of the round, C is playing catch-up, the probability that C wins w can be computed exactly as in section 6.0.1:

$$w(x) = c_1 \left(\frac{p}{1-p}\right)^{x+1} + c_2$$

where c_1, c_2 are determined by (6.0.12). So we have the equation

$$e_1(x) - pe_1(x-1) - (1-p)e_1(x+1) = \varepsilon_1 \left(c_1 \left(\frac{p}{1-p}\right)^{x+1} + c_2\right) \tag{7.1.8}$$

which is quite similar to the equation (6.0.14).

As before

$$\begin{aligned} y_1 &= -\varepsilon_1 \frac{c_2}{1-2p} x \\ y_2 &= \frac{\varepsilon_1 c_1 x}{1-2p} \left(\frac{p}{1-p}\right)^{x+1} \end{aligned}$$

will solve the equations, respectively,

$$\begin{aligned} y(x) - py(x-1) - (1-p)\,y(x+1) &= \varepsilon_1 c_2 \\ y(x) - py(x-1) - (1-p)\,y(x+1) &= \varepsilon_1 c_1 \left(\frac{p}{1-p}\right)^{x+1}. \end{aligned}$$

So the general solution is

$$y(x) = \tilde{c}_1 \left(\frac{p}{1-p}\right)^{x} + \tilde{c}_2 - \varepsilon_1 \frac{c_2}{1-2p} x + \frac{\varepsilon_1 c_1 x}{1-2p} \left(\frac{p}{1-p}\right)^{x+1}. \tag{7.1.9}$$

We can solve this with boundary conditions

$$\begin{aligned} e_1(-1) &= 0 \\ e_1(k) &= 0. \end{aligned}$$

Again recall why the zero boundary conditions are there: Because because we are solving for $s = 0$ in the second state variable, there will be nothing gained or lost for this mining pool, so they are suitable.

Solving,

$$\begin{aligned} \tilde{c}_1 \left(\frac{p}{1-p}\right)^{-1} + \tilde{c}_2 + \varepsilon_1 \frac{c_2}{1-2p} - \frac{\varepsilon_1 c_1}{1-2p} &= 0 \\ \tilde{c}_1 \left(\frac{p}{1-p}\right)^{k} + \tilde{c}_2 - \varepsilon_1 \frac{c_2}{1-2p} k + \frac{\varepsilon_1 c_1}{1-2p} k \left(\frac{p}{1-p}\right)^{k+1} &= 0 \end{aligned}$$

or

$$\begin{pmatrix} m^{-1} & 1 \\ m^k & 1 \end{pmatrix} \begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \end{pmatrix} = \frac{\varepsilon_1}{1-2p} \begin{pmatrix} 1 & -1 \\ -km^{k+1} & k \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}.$$

(We are going to proceed attempting to write most expressions in terms of matrices - this will make using a computer with linear algebra capabilities much simpler to code and will simplify presentation.) Solving as before, we get

$$\begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \end{pmatrix} = \frac{m}{1-m^{k+1}} \frac{\varepsilon_1}{1-2p} \begin{pmatrix} 1 & -1 \\ -m^k & m^{-1} \end{pmatrix} \begin{pmatrix} 1 & -1 \\ -km^{k+1} & k \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}. \tag{7.1.10}$$

We can plug this into get the solution

$$e_1(x) = \tilde{c}_1 m^x + \tilde{c}_2 - \varepsilon_1 \frac{c_2}{(1-2p)} x + \varepsilon_1 \frac{c_1}{1-2p} x m^{x+1}.$$

Now we are interested in the values

$$\mathbb{E}[L_1 \mid (1,0)] = e_1(1) = \tilde{c}_1 m + \tilde{c}_2 - \varepsilon_1 \frac{c_2}{(1-2p)} + \varepsilon_1 \frac{c_1}{1-2p} m^2, \tag{7.1.11}$$
$$\mathbb{E}[L_1 \mid (1,1)] = e_1(1) + w(1).$$

We have determined that the expected number of blocks that are lost in a given round, recalling (7.1.5) is

$$\begin{aligned} \mathbb{E}[L_1] &= \frac{\varepsilon_1}{1-p}\left(e_1(1) + w(1)\right) + \left(1 - \frac{\varepsilon_1}{1-p}\right) e_1(1) \\ &= e_1(1) + \frac{\varepsilon_1}{1-p} w(1). \end{aligned}$$

Now we aren't quite done - we wanted to know what the expected number of blocks that are lost, given the fact that C won the round. To compute this use

$$\mathbb{E}[L_1] = w(1)\mathbb{E}[L_1 \mid \text{C wins}] + (1 - w(1))\mathbb{E}[L_1 \mid \text{C loses}]$$

Because

$$\mathbb{E}[L_1 \mid \text{C loses}] = 0$$

we can then determine that

$$\begin{aligned} \mathbb{E}[L_1 \mid \text{C wins}] &= \frac{\mathbb{E}[L_1]}{w(1)} \\ &= \frac{e_1(1)}{w(1)} + \frac{\varepsilon_1}{1-p}. \end{aligned} \tag{7.1.12}$$

Thus we determine

$$\tau = \frac{e_1(1)}{w(1)} + \frac{\varepsilon_1}{1-p}.$$

Now σ is easier; it is the expected share of the additional reward. This is easily seen to be

$$\sigma = \frac{\varepsilon_1}{1-p} \lambda$$

as once the round starts the odds of winning don't depend on which pool mined the offending block. So the payoff to Pool 1 is

$$\begin{aligned}
&\frac{\varepsilon_1}{1-p}\lambda - \left(\frac{e_1(1)}{w(1)} + \frac{\varepsilon_1}{1-p}\right)\frac{w(1)}{1-w(1)} \\
&= \frac{\varepsilon_1}{1-p}\left(\lambda - \frac{w(1)}{1-w(1)}\right) - \frac{e(1)}{1-w(1)} \qquad (7.1.13)
\end{aligned}$$

where $e_1(1)$ is determined by (7.1.11), (7.1.10).

Summary To summarize:

1. Solve c_1, c_2 via (6.0.12).
2. Solve $\tilde{c}_1, \tilde{c}_2$ via (7.1.10)
3. Solve $e_1(1)$ using (7.1.11)
4. Note that $w(1) = c_1 m + c_2$
5. Harvest the result in (7.1.13). This is $\mathbb{E}[\lambda W_1 - L_1]$ in the (Non-compliant, Non-compliant) strategy regime.

The computation for Pool 2 is identical. One only needs to modify 3) to incorporate the different value ε_2.

7.1.4 Case 2 (Non-compliant, Mildly Compliant). Pool 1 is non-compliant, Pool 2 is mildly compliant.

We will begin by attempting to compute $w(x)$. However, there is a wrinkle that prohibits us from using exactly the same calculus we have been using up to this point and simply plug into (6.0.12). These equations are "broken" by the fact that the strategy, and hence the transition probabilities change, in the intermediate states. So we need to perform some slightly more involved computations in order to solve the problem.

Broken difference equations

We begin with the most basic computation, the probability $w(x)$ that C wins the round, given we are at state x. The equations are different than previously: In particular, letting

$$p^* = p + \varepsilon_2$$

we have

$$w(0) = p^* w(-1) + (1 - p^*) w(1). \qquad (7.1.14)$$

This encodes the fact that when $i = 0$, Pool 2's hashrate of ε_2 is joining the censors (hence p^* and not p.) However, for integers $x \geq 1$, Pool 2 returns to mine on the true chain tip, and we have

$$w(x) = pw(x-1) + (1-p)w(x+1). \tag{7.1.15}$$

We solve the broken difference equation using somewhat of a cheat: Look at the second equation (7.1.15). This holds for $x > 0$, and only excludes the point 0, so we may think of (7.1.14) as a single relation that needs to be satisfied, and solve the problem on $[0, k]$ considering this single relation and one degree of flexibility built into the boundary value problem. Note that this works for when there is only one relation in the set of relations (7.1.14).

To begin, because $w(-1) = 1$ we have from (7.1.14)

$$w(0) = p^* + (1-p^*)w(1). \tag{7.1.16}$$

Now if we solve the boundary value problem for the equation (7.1.15) with (for b TBD)

$$\begin{aligned} w(0) &= b \\ w(k) &= 0 \end{aligned}$$

then necessarily from (7.1.16) we see

$$w(1) = \frac{b - p^*}{1 - p^*}.$$

(We have skipped the step of solving the first equation (7.1.14) as an equation: Knowing $w(-1) = 1$, specifying $w(0) = b$, we know what $w(1)$ must be.) So now, we may solve the equation (7.1.15) with three values

$$\begin{aligned} w(0) &= b \\ w(1) &= \frac{b - p^*}{1 - p^*} \\ w(k) &= 0. \end{aligned}$$

This extra parameter b allows us room to put in the third condition, and we expect to get a unique solution.

As the equation (7.1.15) is the same equation we have seen before, we recall that

$$w(x) = c_1 m^x + c_2.$$

Thus the boundary conditions above become

$$\begin{aligned} c_1 + c_2 &= b \\ c_1 m + c_2 &= \frac{b - p^*}{1 - p^*} \\ c_1 m^k + c_2 &= 0 \end{aligned}$$

or when written as a matrix:

$$\begin{pmatrix} 1 & 1 \\ m & 1 \\ m^k & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} b \\ \frac{b-p^*}{(1-p^*)} \\ 0 \end{pmatrix}. \tag{7.1.17}$$

Now we augment up the matrix in order to solve the problem completely using linear algebra. Notice that

$$\begin{pmatrix} b \\ \frac{b-p^*}{1-p^*} \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{1-p^*} \\ 0 \end{pmatrix} b + \begin{pmatrix} 0 \\ \frac{-p^*}{1-p^*} \\ 0 \end{pmatrix}$$

So (7.1.17) is equivalent to

$$\begin{pmatrix} 1 & 1 \\ m & 1 \\ m^k & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} - \begin{pmatrix} 1 \\ \frac{1}{1-p^*} \\ 0 \end{pmatrix} b = \begin{pmatrix} 0 \\ \frac{-p^*}{1-p^*} \\ 0 \end{pmatrix}$$

or

$$\begin{pmatrix} 1 & 1 & -1 \\ m & 1 & \frac{-1}{1-p^*} \\ m^k & 1 & 0 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ b \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{-p^*}{1-p^*} \\ 0 \end{pmatrix}.$$

Now, in order to solve for c_1 and c_2 (and b incidentally) we need to invert this 3×3 matrix,

$$\begin{pmatrix} c_1 \\ c_2 \\ b \end{pmatrix} = \begin{pmatrix} 1 & 1 & -1 \\ m & 1 & \frac{-1}{1-p^*} \\ m^k & 1 & 0 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ \frac{-p^*}{1-p^*} \\ 0 \end{pmatrix}. \tag{7.1.18}$$

If we are performing these computations via a computer, this method may be preferable.

Now that we have c_1 and c_2 we can compute our new function $w(x)$, now given as a multipart function:

$$\begin{aligned} w(-1) &= 1 \\ w(x) &= c_1 m^x + c_2 \text{ for } x \geq 0. \end{aligned}$$

Remark 7.1.2. If we had cut off at say $i = 2$, the computation would have been more complicated. Another method would be to solve both equations, but with "buckled" boundary conditions that overlap at two points, and a free parameter (such a b above) specified for one of the boundary conditions.

Expected value for Pool 1

Now can we use this same strategy to compute expected values. To begin, we have

$$\mathbb{E}[L_1] = \frac{\varepsilon_1}{q+\varepsilon_1}\mathbb{E}[L_1 \mid (1,1)] + \frac{q}{q+\varepsilon_1}\mathbb{E}[L_1 \mid (1,0)]. \tag{7.1.19}$$

But then, using $e_1(x,s)$ for short-hand we have a single relation describing the transitions at state $(0,s)$

$$e_1(0,s) = p^* e_1(-1,s) + \varepsilon_1 e_1(1,s+1) + qe(1,s).$$

As in (7.1.7)

$$e_1(1,s+1) = e_1(1,s) + w(1) \tag{7.1.20}$$

so

$$e_1(0,s) = p^* e_1(-1,s) + \varepsilon_1\left(e_1(1,s) + w(1)\right) + qe_1(1,s)$$

or

$$e_1(0,s) - p^* e_1(-1,s) - (1-p^*)e_1(1,s) = \varepsilon_1 w(1). \tag{7.1.21}$$

As above when we solved the broken equation for $w(x)$, this relation will be plugged in directly to create a three parameter boundary value problem when solving for $e_1(x)$.

For $x \geq 1$

$$e_1(x,s) = pe(x-1,s) + \varepsilon_1 e_1(x+1,s+1) + (q+\varepsilon_2)\, e_1(x+1,s)$$

which gives a familiar equation

$$e_1(x,s) - e_1(x-1,s) - (1-p)e_1(x+1,s) = \varepsilon_1 w(x+1). \tag{7.1.22}$$

The general strategy for this type of broken system of non-homogeneous equation is similar to as the homogeneous equation. We will set up the general solution for (7.1.22) and then use (7.1.21) to set up a third condition.

Solving (7.1.8) as before to get the general solution (7.1.9)

$$y(x) = \tilde{c}_1\left(\frac{p}{1-p}\right)^x + \tilde{c}_2 - \varepsilon_1\frac{c_2}{1-2p}x + \frac{\varepsilon_1 c_1 x}{1-2p}\left(\frac{p}{1-p}\right)^{x+1}. \tag{7.1.23}$$

where c_1, c_2 are obtained by (7.1.18). Setting boundary condition

$$\begin{aligned} e_1(0) &= b_1 \\ e_1(k) &= 0 \end{aligned}$$

and then bringing in (7.1.21):

$$e_1(1) = \frac{b_1 - \varepsilon_1 w(1)}{1-p^*}. \tag{7.1.24}$$

We are trying to solve for $\tilde{c}_1, \tilde{c}_2$, so we plug the values for e_1 in the three previous expressions to get

$$\begin{aligned}\tilde{c}_1 + \tilde{c}_2 &= b_1\\ \tilde{c}_1 m + \tilde{c}_2 - \varepsilon_1 \frac{c_2}{1-2p} + \frac{\varepsilon_1 c_1}{1-2p} m^2 &= \frac{b_1 - \varepsilon_1 w(1)}{1-p^*}\\ \tilde{c}_1 m^k + \tilde{c}_2 - \varepsilon_1 \frac{c_2}{1-2p} k + \frac{\varepsilon_1 c_1 k}{1-2p} m^{k+1} &= 0\end{aligned}$$

which tidies up into matrix notation as

$$\begin{aligned}\begin{pmatrix} 1 & 1 & -1 \\ m & 1 & -\frac{1}{1-p^*} \\ m^k & 1 & 0 \end{pmatrix} \begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \\ b_1 \end{pmatrix} &= \begin{pmatrix} 0 \\ \frac{-\varepsilon_1 w(1)}{1-p^*} + \varepsilon_1 \frac{c_2}{1-2p} - \frac{\varepsilon_1 c_1}{1-2p} m^2 \\ \varepsilon_1 \frac{c_2}{1-2p} k - \frac{\varepsilon_1 c_1 k}{1-2p} m^{k+1} \end{pmatrix} \qquad (7.1.25)\\ &= \varepsilon_1 \begin{pmatrix} 0 & 0 & 0 \\ \frac{-w(1)}{1-p^*} & -\frac{m^2}{1-2p} & \frac{1}{1-2p} \\ 0 & -\frac{k}{1-2p} m^{k+1} & \frac{k}{1-2p} \end{pmatrix} \begin{pmatrix} 1 \\ c_1 \\ c_2 \end{pmatrix}.\end{aligned}$$

We can solve for the coefficients $\tilde{c}_1, \tilde{c}_2$ using linear algebra. Once we have done this, plug back into (7.1.19), (7.1.20)

$$\begin{aligned}\mathbb{E}[L_1] &= \frac{\varepsilon_1}{q+\varepsilon_1} \left(e_1(1) + w(1)\right) + \frac{q}{q+\varepsilon_1} e(1)\\ &= e_1(1) + \frac{\varepsilon_1}{q+\varepsilon_1} w(1).\end{aligned}$$

As before (7.1.12) we can then determine that

$$\begin{aligned}\tau &= \frac{e_1(1)}{w(1)} + \frac{\varepsilon_1}{q+\varepsilon_1}\\ \sigma &= \frac{\varepsilon_1}{q+\varepsilon_1} \lambda.\end{aligned}$$

Once the round starts, the odds of winning don't depend on which pool mined the offending block. So the payoff to Pool 1 is

$$\begin{aligned}&\frac{\varepsilon_1}{q+\varepsilon_1} \lambda - \left(\frac{e_1(1)}{w(1)} + \frac{\varepsilon_1}{q+\varepsilon_1}\right) \frac{w(1)}{1-w(1)}\\ &= \frac{\varepsilon_1}{q+\varepsilon_1} \left(\lambda - \frac{w(1)}{1-w(1)}\right) - \frac{e_1(1)}{1-w(1)}. \qquad (7.1.26)\end{aligned}$$

This gives the expected value to Pool 1 in the second case.

Summary

1. Solve for w, that is find coefficients c_1, c_2 using (7.1.18).

2. Solve for $\tilde{c}_1, \tilde{c}_2, b_1$ using (7.1.25) and $w(1) = c_1 m + c_2$.
3. Solve for $e_1(1)$ using (7.1.24).
4. Harvest $\mathbb{E}[\lambda W_1 - L_1]$ using (7.1.26).

We note that because we are using b_1 to solve for the value $e_1(1)$ we end up not needing the information given by $\tilde{c}_1$ and $\tilde{c}_2$ in this particular computation.

Expected value for Pool 2

We have already computed $w(x)$ in this strategy regime. In setting up the computation of the outcome for Pool 2, we now let the second state variable denote the number of blocks that Pool 2 has mined. Note that because the Pool 2 is mining with the censorial pool only when the chains are equal length, Pool 2 has no risk of censorial blocks being orphaned. The risk to Pool 2 is realized only if Pool 2 adds to a NC chain that is already 1 ahead, and this chain ends up losing. (An obvious tweak to this strategy would be only switch to a less compliant strategy after the event that Pool 2 has mined a non-compliant block as the current strategy has the unlikely but positive probability that the chain will mine a block on the NC chain, and then end up breaking a tie in favor of C, hence orphaning their own block.)

Because the pool isn't interested in mining the objectionable transaction itself, we start with

$$\mathbb{E}[L_2] = \mathbb{E}[L_2 \mid (1, 0)] \tag{7.1.27}$$

as the initial state necessarily involves either Pool 1 or NC having just mined the block. Now we have

$$\mathbb{E}[L_2 \mid (0, s)] = (p + \varepsilon_2)\,\mathbb{E}[L_2 \mid (-1, s)] + (1 - p - \varepsilon_2)\mathbb{E}[L_2 \mid (1, s)]$$

and for $x \geq 1$

$$\mathbb{E}[L_2 \mid (x, s)] = p\mathbb{E}[L_2 \mid (x-1, s)] + (1-p-\varepsilon_2)\mathbb{E}[L_2 \mid (x+1, s)] + \varepsilon_2\mathbb{E}[L_2 \mid (x+1, s+1)].$$

Also as before

$$\mathbb{E}[L_2 \mid (x, s + 1)] = \mathbb{E}[L_2 \mid (x, s)] + w(x)$$

So the second relation (using $e_2(x) = \mathbb{E}[L_2 \mid (x, 0)]$) becomes

$$e_2(x) - pe_2(x - 1) - (1 - p)e_2(x + 1) = \varepsilon_2 w(x + 1) \text{ for } x > 0.$$

The function $w(x)$ has been computed, with coefficients determined by (7.1.18). The analysis to determine $e_2(x)$ is very similar to the analysis line following (7.1.23). With general solution

$$y(x) = \tilde{c}_1\left(\frac{p}{1-p}\right)^x + \tilde{c}_2 - \varepsilon_2\frac{c_2}{1-2p}x + \frac{\varepsilon_2 c_1 x}{1-2p}\left(\frac{p}{1-p}\right)^{x+1} \tag{7.1.28}$$

We solve for boundary condition

$$\begin{aligned} e_2(0) &= b_2 \\ e_2(k) &= 0 \end{aligned} \tag{7.1.29}$$

and the additional condition with free parameter b_2

$$e_2(1) = \frac{b_2}{1-p^*}. \tag{7.1.30}$$

which follows from (7.1.29) and

$$e_2(-1) = 0$$

$$e_2(0) - p^* e_2(-1) - (1-p^*)e_2(1) = 0 \tag{7.1.31}$$

which differs from (7.1.21) because there is no $w(1)$ term on the right hand side.

We are trying to solve for $\tilde{c}_1, \tilde{c}_2$, so we plug the boundary values in

$$\begin{aligned} \tilde{c}_1 + \tilde{c}_2 - b_2 &= 0 \\ \tilde{c}_1 m + \tilde{c}_2 - \frac{1}{1-p^*} b_2 &= \varepsilon_2 \frac{c_2}{1-2p} - \frac{\varepsilon_2 c_1}{1-2p} m^2 \\ \tilde{c}_1 m^k + \tilde{c}_2 &= \varepsilon_2 \frac{c_2}{1-2p} k - \frac{\varepsilon_2 c_1 k}{1-2p} m^{k+1} \end{aligned}$$

and convert to matrix notation as

$$\begin{aligned} \begin{pmatrix} 1 & 1 & -1 \\ m & 1 & -\frac{1}{1-p^*} \\ m^k & 1 & 0 \end{pmatrix} \begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \\ b_2 \end{pmatrix} &= \begin{pmatrix} 0 \\ \varepsilon_2 \frac{c_2}{1-2p} - \frac{\varepsilon_2 c_1}{1-2p} m^2 \\ \varepsilon_2 \frac{c_2}{1-2p} k - \frac{\varepsilon_2 c_1 k}{1-2p} m^{k+1} \end{pmatrix} \\ &= \frac{\varepsilon_2}{1-2p} \begin{pmatrix} 0 & 0 \\ -m^2 & 1 \\ -km^{k+1} & k \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}. \end{aligned} \tag{7.1.32}$$

Inverting the first matrix, we can solve for the coefficients $\tilde{c}_1, \tilde{c}_2$ and b_2. Once we have done this, plug back into

$$\mathbb{E}[L_2] = e_2(1) = \frac{b_2}{1-p^*}. \tag{7.1.33}$$

As before we can then determine that

$$\begin{aligned} \tau &= \frac{e_2(1)}{w(1)} \\ \sigma &= 0 \end{aligned}$$

So the total expectation to Pool 2 is

$$-\frac{e_2(1)}{1-w(1)} = -\frac{b_2}{(1-w(1))\,(1-p^*)}. \tag{7.1.34}$$

Summary

1. Solve for w, that is find coefficients c_1, c_2 using (7.1.18).
2. Solve for $\tilde{c}_1, \tilde{c}_2, b_2$ using (7.1.32) and $w(1) = c_1 m + c_2$.
3. Solve for $e_2(1)$ using (7.1.30).
4. Harvest $\mathbb{E}[\lambda W_2 - L_2]$ using (7.1.34)

7.1.5 Case 3 Both Pools Mildly compliant

This will be almost identical to the computation in section 7.1.4. This time we take

$$p^* = p + \varepsilon_1 + \varepsilon_2.$$

Because neither pool is interested in mining the objectionable transaction itself, we start with

$$\mathbb{E}[L_i] = \mathbb{E}[L_i \mid (1,0)]$$

as the initial state is that only NC has mined the block. Now we have

$$\mathbb{E}[L_i \mid (0,s)] = p^* \mathbb{E}[L_i \mid (-1,s)] + (1-p^*)\mathbb{E}[L_i \mid (1,s)]$$

for $x = 0$ and

$$\mathbb{E}[L_i \mid (x,s)] = p\mathbb{E}[L_i \mid (x-1,s)] + (1-p-\varepsilon_i)\mathbb{E}[L_i \mid (x+1,s)] + \varepsilon_i \mathbb{E}[L_i \mid (x+1,s+1)]$$

for $x \geq 1$. Also as before

$$\mathbb{E}[L_i \mid (x,s+1)] = \mathbb{E}[L_i \mid (x,s)] + w(x)$$

So the second relation becomes

$$e_i(x) - pe_i(x-1) - (1-p)e_i(x+1) = \varepsilon_i w(x+1) \quad \text{for } x \geq 1$$

The function $w(x)$ is determined from coefficients given by (7.1.18). Setting boundary condition

$$\begin{aligned} e_i(0) &= b_i^* \\ e_i(1) &= \frac{b_i^*}{1-p^*} \\ e_i(k) &= 0 \end{aligned} \tag{7.1.35}$$

Which converts to matrix notation as

$$\begin{pmatrix} 1 & 1 & -1 \\ m & 1 & -\frac{1}{1-p^*} \\ m^k & 1 & 0 \end{pmatrix} \begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \\ b_i^* \end{pmatrix} = \varepsilon_i \begin{pmatrix} 0 & 0 \\ -\frac{m^2}{1-2p} & \frac{1}{1-2p} \\ -\frac{k}{1-2p} m^{k+1} & \frac{k}{1-2p} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \tag{7.1.36}$$

Inverting the matrix, we can solve for the coefficients $\tilde{c}_1, \tilde{c}_2$. Once we have done this

$$\mathbb{E}[L_i] = e_i(1) \tag{7.1.37}$$

So the total expectation to Pool 2 is

$$\begin{aligned} & -\frac{e_i(1)}{1-w(1)} && (7.1.38) \\ &= -\frac{b_i^*}{(1-w(1))\,(1-p^*)} && (7.1.39) \end{aligned}$$

Inspecting (7.1.36) we see that

$$\frac{b_1^*}{b_2^*} = \frac{\varepsilon_1}{\varepsilon_2}.$$

Summary

1. Solve for w, that is find coefficients c_1, c_2 using (7.1.18).
2. For either ε_1 or ε_2, solve for $\tilde{c}_1, \tilde{c}_2, b$ using (7.1.36) and $w(1) = c_1 m + c_2$.
3. Solve for $e_i(1)$ using (7.1.35).
4. Harvest $\mathbb{E}[\lambda W_2 - L_2]$ using (7.1.38).

7.1.6 Analysis: Low fee regime

We begin by looking at situations in which the fee paid by the censored party is relatively small. As of September 2021, the block reward is about 6.25 BTC @ \$50,000 = \$312,500. So if we assume the censoring party is willing to party with \$312.50 in order to process the transaction, this will amount to $\lambda = 0.001$.

Normalized Tables

We will used "normalized tables", which divide the expectations by the hash rate of each of the pools, as follows,

Popl 2 strategy / Pool 1 strategy	Non-compliant	Mildly Compliant
Non-compliant	$\frac{\mathbb{E}[\lambda W_1 - L_2]}{\varepsilon_1}, \frac{\mathbb{E}[\lambda W_2 - L_2]}{\varepsilon_2}$	
Mildly Compliant		

Notice that in general, re-normalization of any agent's payoff will not effect strategic considerations. Dominant strategy will remain dominant and so forth. Agents are attempting to maximize their own outcomes, not their outcomes relative to someone else's outcomes. Notice that hashing fraction of pool represents the expected block rewards per ten minutes of mining. So the payoff number given in the normalized table is given in unit of expected payoff for ten minutes of mining for that given pool.

To illustrate this, we start with $p = 0.2, \varepsilon_1 = 0.15, \varepsilon_2 = 0.1$. A censoring pool has 20% of hashing power, and two other rational pools, with hashing powers 15% and and 10% are deciding between non-compliant and mildly compliant strategies. We start with $\lambda = 0.001$ quite small and $k = 4$.

	Non-compliant	Mildly Compliant
Non-compliant	$(-0.01944, -0.01296)$	$(-0.03154, -0.00416)$
Mildly Compliant	$(-0.00719, -0.0256)$	$(-0.00897, -0.00598)$

(Unnormalized payoff: $p = 0.2, \varepsilon_1 = 0.15$, $\varepsilon_2 = 0.1, \lambda = 0.001, k = 4$)

For comparison, we show the normalized payoff table:

	Non-compliant	Mildly Compliant
Non-compliant	$(-0.1296, -0.1296)$	$(-0.03154, -0.0416)$
Mildly Compliant	$(-0.0479, -0.256)$	$(-0.0598, -0.0598)$

(Normalized payoff: $p = 0.2, \varepsilon_1 = 0.15$, $\varepsilon_2 = 0.1, \lambda = 0.001, k = 4$)

Observe this normalized table has some symmetry along the diagonal: It represents that for a censoring of censoring power 20% and a transaction only offering 0.1% of a block reward as compensation, if both miners use a non-compliant strategy, they are each expected to lose about 13% of their expected reward over ten minutes of typical mining. (The game itself could last several blocks.) By comparison, if both mining pools adopt a mildly compliant strategy they expect to lose about 6% of their expectations over ten minutes.

We observe quite clearly the type of game this represents. Both players will play their dominant strategy, and this will maximize the payoff to both players collectively. This is called a **Deadlock game**. A deadlock game is a game in which each play has a clearly dominant strategy, and this is mutually beneficial. Typically such games are less interesting, because players choose the strategy that benefits others represented in the payoff table out of their own self-interest. Of course, the party whose interest is not represented by this table is the party sending the transaction. However, once they have sent the transaction and offered a fee, they are not involved in the mining game.

Both pools are motivated to use a mildly compliant strategy, at least when given the choice between mildly compliant and non-compliant. The strategy space is a quite large so we cannot conclude that this strategy is a dominant among all possible strategies, only dominant when compared against a non-compliant strategy.

A pool of size 20% maybe considered a large censoring pool, as it represents already 40% of the way towards full control of the network. How effective will censorship with lower combinations of hash power? Reducing this censor's power to 5% shows a marked difference:

	Noncompliant	Mildly Compliant
Noncompliant	$(-0.00219, -0.00219)$	$(-0.0096, -0.0005)$
Mildly Compliant	$(-0.00067, -0.0139)$	$(-0.0010, -0.0010)$

(Normalized payoff: $p = 0.05, \varepsilon_1 = 0.15,\ \varepsilon_2 = 0.1, \lambda = 0.001, k = 4$)

Note that while the dynamics of the game are still essentially the same, for low hashrate of the censor the motivations towards rational pools may be approaching indifference: Lose 0.2% of your expected rewards for 10 minutes of mining versus lose 0.1%, this is equivalent to a difference of about 6 seconds of mining.

What if we consider larger rational pools?

	Non-compliant	Mildly Compliant
Non-compliant	$(-0.00219, -0.00219)$	$(-0.00287, -0.000203)$
Mildly Compliant	$(-0.00182, -0.0691)$	$(-0.00185, -0.00185)$

(Normalized payoff: $p = 0.05, \varepsilon_1 = 0.5,\ \varepsilon_2 = 0.01, \lambda = 0.001, k = 4$)

Still, the incentive to comply is still rather small, approaching indifference.

Now what about when the C pool is larger, say 35%? First we consider small-sized rational pools.

	Non-compliant	Mildly Compliant
Non-compliant	$(-1.204, -1.204)$	$(-1.204, -0.423)$
Mildly Compliant	$(-0.423, -1.204)$	$(-0.423, -0.423)$

(Normalized payoff: $p = 0.35, \varepsilon_1 = 0.0001,\ \varepsilon_2 = 0.0001, \lambda = 0.001, k = 4$)

Perhaps not surprisingly, when the rational pools are small, the payoff tables are nearly independent of the other pools behavior (they actually are different, but the difference is small than the precision offered in the tables. For either pool there is a distinct incentive (about 8 minutes of mining) to switch to a mildly compliant strategy.

For comparison

	Non-compliant	Mildly Compliant
Non-compliant	$(-1.204, -1.204)$	$(-1.612, -0.5065)$
Mildly Compliant	$(-0.5795, -2.126)$	$(-0.6438, -0.6438)$

(Normalized payoff: $p = 0.35, \varepsilon_1 = 0.2,\ \varepsilon_2 = 0.1, \lambda = 0.001, k = 4$)

Unsurprisingly, the incentive becomes more certain as the censorial pool approaches 50%

	Non-compliant	Mildly Compliant
Non-compliant	$(-5.565, -5.565)$	$(-6.196, -2.321)$
Mildly Compliant	$(-2.321, -6.196)$	$(-2.416, -2.416)$

(Normalized payoff: $p = 0.47, \varepsilon_1 = 0.05,\ \varepsilon_2 = 0.05, \lambda = 0.001, k = 4$)

Here a decision to resist the censors will result in an additional expected loss of over twenty minutes of mining. This may still seem small. When the censorial pool extends k (which will make more sense as $p \to 0.5$) there is significant increase in expected loss.

	Non-compliant	Mildly Compliant
Non-compliant	$(-29.224, -29.224)$	$(-30.952, -21.571)$
Mildly Compliant	$(-21.571, -30.952)$	$(-22.096, -22.096)$

(Normalized payoff: $p = 0.47, \varepsilon_1 = 0.05, \varepsilon_2 = 0.05, \lambda = 0.001, k = 10$)

.

7.1.7 Higher fee regime

The game changes if the party sending the offending transaction is willing to pay. In the following, we take

$$\lambda = 0.16.$$

This represents one bitcoin when the block reward is 6.25 bitcoins.

	Non-compliant	Mildly Compliant
Non-compliant	$(0.06912, 0.06912)$	$(0.04383, -0.03525)$
Mildly Compliant	$(-0.03525, 0.04383)$	$(-0.04167, -0.04167)$

(Normalized payoff: $p = 0.2, \varepsilon_1 = 0.05, \varepsilon_2 = 0.05, \lambda = 0.16, k = 4$)

The tables have turned. The non-compliant strategy is now dominant, at least when $p = 0.2$. Increasing to $p = 0.25$, it's more interesting

	Non-compliant	Mildly Compliant
Non-compliant	$(-0.07725, -0.07725)$	$(-0.1306, -0.09205)$
Mildly Compliant	$(-0.09205, -0.1306)$	$(-0.1052, -0.1052)$

(Normalized payoff: $p = 0.25, \varepsilon_1 = 0.05, \varepsilon_2 = 0.05, \lambda = 0.16, k = 4$)

Notice that this game has no dominant strategies available, and has two Nash equilibria. This game resembles a Stag Hunt. Making the first pool much larger we again have a situation when non-compliant is dominant for both.

	Non-compliant	Mildly Compliant
Non-compliant	$(-0.07725, -0.07725)$	$(-0.1306, -0.09205)$
Mildly Compliant	$(-0.2028, -1.463)$	$(-0.2118, -0.2118)$

(Normalized payoff: $p = 0.25, \varepsilon_1 = 0.05, \varepsilon_2 = 0.55, \lambda = 0.16, k = 4$)

Playing with parameters, one can find asymmetrical examples:

	Non-compliant	Mildly Compliant
Non-compliant	$(-0.1039, -0.1039)$	$(-0.1593, -0.09205)$
Mildly Compliant	$(-0.2028, -1.563)$	$(-0.2118, -0.2118)$

(Normalized payoff: $p = 0.25, \varepsilon_1 = 0.05$, $\varepsilon_2 = 0.55, \lambda = 0.14, k = 4$)

Here non-compliant is dominant for the larger pool but there is no dominant strategy for the smaller pool. However, because the smaller pool can assume that the larger pool is using it's dominant strategy (non-compliant), they can choose their best strategy given this information. So the second pool will choose mildly compliant, and the resulting Nash Equlibrium will be off-diagonal.

7.1.8 Conclusion

In terms of pure game theoretical considerations, it is possible for a pool with censorial objective to change the incentives landscape so that rational pools are motivated to join the censors. However, from the tables above if the pool controls a smaller portion, say 5% of the hashrate, the motivations are likely to not come into play, unless there is extremely low profit margins for the miners. A censorial pool cannot expect to win all their battles, but the goal would be to cause enough losses over time so the rational miners would join them in increasing number. The effect is to grind down the non-compliant pools making non-compliant mining on average less profitable.

There are many other possible strategies that can be explored, and this becomes much more interesting when profitability of the miners is taken into consideration. We are not considering the possibility that non-compliant pools will punish those that join compliant efforts by breaking consensus protocols to mine non-compliant blocks, even at a loss. But this is always possible.

Neither are we considering competition for blockspace. A computation (more favorable to the censors) would subtract from λ the average fee obtainable for the blockspace, according to the market rates. In 2021, this might not be a large difference, but as the block reward decreases over the next 40 years, this modified computation will become a more significant.

Chapter 8

Selfish mining

8.1 Markov Chains

8.2 Dyck Words and Catalan numbers

Chapter 9

Block reward decay

Chapter 10

Nation-state 51% attacks

Chapter 11

Miner collusion

11.0.1 What can colluding pools do?

11.0.2 What can colluding pools not do?

Chapter 12

More on proof of stake

Chapter 13

High stakes games

13.1 Nash Bargaining

13.2 Rubinstein games

13.3 Use of blockchain for mixed strategies and committment

Bibliography

[GP2020] Grunspan, Cyril. and Pérez-Marco, Ricard, *The mathematics of Bitcoin*, Eur. Math. Soc. Newsl. **115** (2020), 31–37

[GP2018] Grunspan, Cyril. and Pérez-Marco, Ricardo, *Double Spend Races*, Int. J. Theor. Appl. Finance 21. **8** (2018)

[Keynes1936] Keynes, John Maynard, *The general theory of employment, interest and money.* London :Macmillan, (1936).

[Lebl2014] Lebl, Jiri. *Notes on Diffy Qs: Differential Equations for Engineers*

[Nakamoto2008] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system.* 2008.

[Rosenfeld2014] Rosenfeld, Meni. *Analysis of hashrate-based double spending.*arXiv:1402.2009, 2014

[SS2011] Stein, Elias M., and Rami Shakarchi. *Functional Analysis: Introduction to Further Topics in Analysis.* Princeton University Press, 2011.

Index