Introduction

This paper is limited to the mathematics of reward curves. This paper is based on a series of blog posts from @theoretical. As discussed in this Steemitblog post, we are re-visiting the choice of curve, and potentially considering a non-quadratic curve for curation rewards.

Pool systems

Funding for rewards is allocated via a *pool system.* STEEM enters the pool at a fixed rate based on the limited speed at which new STEEM is printed. When upvotes (downvotes) occur, post rewards are created (destroyed) in the form of *claims* [1] on a pool [2]. A percentage of the post's total claims are each allocated to curators, authors and (as of HF17) beneficiaries.

Claims are *redeemed* against the STEEM in the pool when the posts are paid out. Pre-HF17, the *redemption rate* (STEEM per claim) is determined using what I will dub the *accountant's algorithm* [3]. The accountant's algorithm sets the exchange rate by dividing the pool's assets by outstanding claims. Post-HF17, the redemption rate is determined using a different algorithm, which I will dub the the *manager's algorithm* [4]. The manager's algorithm sets the exchange rate by dividng the pool's assets by the number of total recent claims. The total recent claims are calculated by adding together all claims being redeemed in the current block, and exponentially discounted past claims in all past blocks.

[1] In previous notes, claims have been called `rshares2` or V-shares. The new name "claims" is adopted as of HF17.

[2] As of HF17, two different pools exist: Top-level posts and comment rewards have separate pools.

[3] The accountant's algorithm is based on a concept of "if hypothetically everything was unwound based on a snapshot of the current moment, we know how much everybody should get paid such that there's exactly zero left over."

[4] The manager's algorithm is based on a concept of "we can't know for sure what the future will bring, but we can still set policy by estimating the future based on the recent past."

Voting

In Steem, each user has Steem Power, which determines the effectiveness of an upvote action by the user. Each user also has voting power, which declines with each vote and regenerates over time. Voting power creates a soft limit on the number of upvotes a user can perform. Users also have the option to vote with

less than their full strength (although this option is not shown in the UI for users with less than a certain amount of SP).

For the purposes of this paper, the product of these factors is called *votes*. Votes are the quantity which used to upvote (or downvote) a post. In the code and some previous communications, votes are called *R-shares*.

Reward system dynamics

Since rewards exist in the form of claims on the reward pool, rewards are

Reward can be divided into two components, a *base reward* and a *bonus reward*. The base reward is the reward assigned to a post without any upvotes; the bonus reward is additional reward based on the *reward curve*. Due to the effect of bonus rewards, 5 users voting on a single post will create *more* rewards than the same 5 users voting on different posts.

When it comes to assigning curation rewards:

- The base component of the new curation reward is assigned to the upvoter.
- The bonus component is assigned proportionally to all previous curators.

Here's an example. In this example, Alice, Bob and Charlie each use 1000 votes.

- Alice is the first upvoter with 1,000,000 claims base reward, and no bonus reward. Alice gets 1,000,000 claims.
- Bob is the second upvoter with 1,000,000 claims base reward and 10,000 claims bonus reward. Bob gets 1,000,000 claims and Alice gets 10,000 additional claims.
- Charlie is the third upvoter with 1,000,000 claims base reward and 20,000 claims bonus reward. Charlie gets 1,000,000 claims, Alice and Bob split the remaining 20,000 claims. Since Alice has 1,010,000 claims and Bob has 1,000,000 claims, the split isn't quite equal: Alice gets 10,050 additional claims while Bob gets 9,950 additional claims, the 101:100 ratio of the newly distributed claims being equal to the 101:100 ratio of the existing claims. At the end of Charlie's upvote processing, Alice has 1,020,050 claims; Bob has 1,009,950 claims; Charlie has 1,000,000 claims.

The *reward curve* determines the number of base and bonus claims to be distributed. In this example, the base reward rate is fixed at 1000 claims / vote; the bonus reward rate increases by 10,000 claims for every 1000 votes. The reward curve is hard-coded by the system designers. This is a simple, made-up reward curve for the purposes of showing an example computation with simple, concrete numbers. Later in this paper we will discuss the actual reward curve used by Steem, including the update to this curve upcoming in HF17.

Reward curve design

Let us establish some notation. Let $V(r)$ be the total payout, in claims, that happens once r votes have occurred [1]. There are some obvious design restrictions we can place on $V(r)$:

- $V(r)$ should have positive slope everywhere (additional upvotes are always good).
- Slope of $V(r)$ should never decrease (later upvotes boost reward at least as much as earlier upvotes).
- Base reward is positive and equal to initial slope of $V(r)$ (thus slope never approaches zero).
- A simple and efficient implementation of the function can be programmed.

Initially Steem used a shifted quadratic reward function $f(r) = r^2$, giving $V(r) = f(r+s) - f(s) = (r+s)^2 - s^2 = r^2 + 2rs$ where s is a global parameter hard-coded by the designers. Implementationally this is certainly straightforward and efficient; in some sense, a quadratic function is the simplest possible non-linear function.

The reward function's slope, $V'(r)$, is simply the current claim-per-vote rate at which rewards are issued. Deine the *flux ratio* $\phi(r)$ of a reward curve to be the ratio of the current slope to the base reward rate; mathematically, $\phi(r) = V'(r)/V'(0)$. The design restrictions imply that $\phi(r)$ is increasing and $\phi(r) \geq 1$, with equality only when $r = 0$. A value of $\phi(r) = 1.05$, for example, represents a point on the curve at which bonus rewards equal to 5% of base rewards are issued. In the quadratic case, $\phi(r) = \frac{r+s}{s}$. We see that $\phi(r)$ is linear, unbounded, and $\phi(s) = 2$.

Some amount of nonlinearity in the reward curve is desirable. Specifically, appreciable flux ratio gains should be possible even above the sizes of the largest whales or coalitions.

[1] The choice of letters here is inspired by historical nomenclature: `r` stands for *(R)eward-shares* and the V stands for *(V)alue-shares*. In this paper we consider the R-shares / V-shares naming to be deprecated in favor of votes / claims. We refrain from using the deprecated names outside footnotes.

Bounded-flux-ratio reward curve

An unbounded flux ratio will tend to create rare high payouts. This characteristic was thought to be desirable when the $f(r) = r^2$ curve was selected: High payouts would provide excitement and thrill over potential high-value outcomes to users, and provide organic user stories as raw material for marketing efforts. In practice, user perception of the unbounded flux ratio has been mixed. One argument against an unbounded flux ratio is that overly optimistic expectations based on

rare large payouts, combined with the large amount of up-front effort necessary for comment creation, cause disappointment, disillusionment and frustration for new users.

The exact shape the reward curve should take is a complex, ongoing, multi-faceted debate which perhaps deserves a study of its own. We will not go into further detail about the debate here, but instead turn our attention to bounded-flux-ratio reward curves.

Let us consider possible candidates for bounded-flux-ratio reward curves. If $f(r) = r^2$ increases too fast, one fairly obvious curve to try is $f(r) = r\log(r)$. However, its derivative is unbounded and an integer-only implementation would be slow and unwieldy.

Consider a function $f(r)$ which behaves as r^2 when r is close to zero, but as r when r grows toward infinity. Perhaps $f(r) = rg(r)$ for some function $g(r)$ which is approximately r for small r, but approximately unity for large r. Such a function is $g(r) = \frac{r}{r+1}$, giving $f(r) = \frac{r^2}{r+1}$. To get a reward curve with slope bounded away from zero, similarly to the quadratic case, we shift this curve by some offset α to obtain:

$$\begin{aligned} V(r) &= f(r+\alpha) - f(\alpha) = \frac{(r+\alpha)^2}{r+1} - \frac{\alpha^2}{r+1} \\ &= \frac{r^2 + 2\alpha r}{r+1} \end{aligned}$$

To help us deal with various technical issues in setting the parameters of the curve later, let us re-define the denominator from $r+1$ to an arbitrary linear function $\beta r + \gamma$:

$$V(r) = \frac{r^2 + 2\alpha r}{\beta r + \gamma}$$

Secondary pool mechanics

One of the implementation issues in the previous example is that Charlie's upvote processing results in increases of claims for Alice and Bob. In general, processing each upvote results in increases in the claims held by all previous upvoters of the same post. We would like to create an implementation that does not need to touch the data structure for previous upvotes when a new upvote is created.

> "All problems in computer science can be solved by another level of indirection." – David Wheeler

In the previous example section, claims against a global, long-lived reward pool are directly assigned to upvoters. Call these "primary claims" against the "primary pool." Taking Wheeler's advice, we'll introduce another level of indirection: Instead of assigning primary claims directly to upvoters, we instead direct the primary claims to a per-post/comment *secondary pool* and issue users *secondary claims*. When the post payout is processed, the secondary claims are redeemed against the secondary pool for primary claims. The primary claims are in turn redeemed against the primary pool for STEEM. Secondary claims are referred to as *weight* in the code.

When weight is issued, it has an *exchange rate* which may be obtained by dividing the (primary-claim-denominated) assets of the secondary pool by its (secondary-claim-denominated) liabilities. (Recall the secondary pool's assets are the curation portion of the base/bonus primary claims generated by upvotes; the secondary pool's liabilities are secondary claims (weight) assigned to curators.) This exchange rate allows any quantity of secondary claims to receive a *valuation* in terms of primary claims.

If, instead of distributing primary claims, we distribute secondary claims based on the desired valuation in primary claims, then we've solved the implementation issue: All earlier upvoters will automatically get their correct benefits through an increase in valuation. As long as the per-upvote data structure denominates rewards in secondary claims, we don't need to iterate through all previous users.

The secondary pool also gives us a simple and natural way to deal with something we've ignored to this point, namely, the effect of downvotes on curation rewards. We simply calculate secondary claims ignoring downvotes, then use the downvotes to reduce the primary claim balance of the secondary pool.

Example using secondary pool

Let's work through the same situation as the previous example, only this time we'll use the secondary pool. As you can see, we will get the same numbers as before (ignoring rounding issues). However, this time we simply let the increase in claims for previous users occur implicitly by an exchange rate increase. We only store weights, and recompute valuations only when needed to pay out the post. So when updating for a new upvote, we don't have to actually change any numbers on previous upvotes' records, we just create a record of the new upvote and update the per-post fields representing the secondary pool assets/liabilities.

- Alice is the first upvoter with 1,000,000 claims base reward, and no bonus reward. The secondary pool gets 1,000,000 primary claims, and Alice gets 1,000,000 weight. The exchange rate of weight to claims is unity (1), so Alice's weight has a valuation of 1,000,000 primary claims.
- Bob is the second upvoter with 1,000,000 claims base reward and 10,000 claims bonus reward. The secondary pool gets 1,010,000 primary claims,

and we issue weight to Bob until his weight's valuation equals 1,000,000 primary claims. This occurs when Bob has 990,099 weight.

- After Bob's award, a total of 1,990,099 weight has been issued, backed by 2,010,000 primary claims. The exchange rate of weight to claims has risen to 1.01 claims per weight. At this exchange rate, Alice's weight is now worth 1,010,000 primary claims, and Bob's weight is worth 1,000,000 primary claims.
- Charlie is the third upvoter with 1,000,000 claims base reward and 20,000 claims bonus reward. The secondary pool gets 1,020,000 primary claims, and we issue weight to Charlie until his weight's valuation equals 1,000,000 primary claims. This occurs when Charlie has 980,344 weight.
- After Charlie's award, a total of 2,970,443 weight has been issued, backed by 3,030,000 primary claims. The exchange rate of weight to claims has risen to 1.020050 claims per weight. At this exchange rate, Alice's weight is now worth 1,020,050 primary claims; Bob's weight is now worth 1,009,950 primary claims; and Charlie's weight is worth 1,000,000 primary claims.

Satoshi-by-satoshi computations

A good reward system should not incentivize Sybil attacks: A user should not be able to gain any advantage by creating multiple accounts and pretending to be multiple users. Conversely, a good reward system should not incentivize pools: A group of users should not be able to gain any advantage by combining their stake into a single account and pretending to be a single user.

What this means is that the above dynamics must necessarily be *micro-dynamics*, meaning the math described in the examples must play out on a satoshi-by-satoshi basis, rather than in 1,000,000-claim blocks as shown in the examples. Since virtually all upvotes involve millions or billions of satoshis, directly executing the micro-dynamics in program code is impractical. Instead, we will open up a toolbox of sophisticated mathematics and *calculate* the net effect of the microdynamics, to obtain an efficient macro-level algorithm which allows us to use only a few arithmetic operations per upvote regardless of the number of satoshis participating.

Ultimate indifference

As we have seen in the examples, Alice, Bob and Charlie each initially get 1,000,000 claims. In other words, the initial valuation of the newly issued weight is a global constant which does not vary over a post's lifetime. This constant initial valuation property is called the CIV property.

To more precisely define the CIV property, we will add some technical conditions:

A reward system has the CIV property if the valuation of an infinitesimally small number of new upvotes, measured in primary claims, is constant across posts, not considering the effect of downvotes.

For an alternative perspective on the CIV property, consider an *ultimate upvoter* (UU), a (theoretical) upvoter who has a small number of votes and is always the last person to upvote a post. The UU is presented with a choice of posts to potentially upvote. Under a reward system with the CIV property, the UU is *indifferent* about which post he/she upvotes because he/she will get the same reward for any post. Therefore, the CIV property is also called the Ultimate Indifference Principle (UIP).

Decreasing weight awards

As we have seen in the example, even though Alice, Bob and Charlie all have the same amount of votes, Alice gets more weight than Bob, who gets more weight than Charlie. A system which awards less weight to later upvoters is decreasing weight award (DWA) property.

From the perspective of system designers, the DWA property is desirable. If Alice is deciding which of two posts to upvote, and the two posts seem equally compelling and likely to generate similar payouts, under a DWA system Alice will be incentivized to vote for the post that's been less upvoted. Therefore, DWA ensures a continued incentive to discover undiscovered gems, and limit the rewards that can be gained by "bandwagoning" onto an already-popular post.

Requirements for curation reward implementation

Let's pause a moment and summarize the requirements of the implementation. These requirements are called "principles."

- The satoshi-by-satoshi principle SBSP. All computations are done on a satoshi-by-satoshi basis. In practice, this means we use the mathematical tools of integral calculus. Votes are fungible, homogenous, and arbitrarily finely divisible. If Alice and Bob both on the same post (and we neglect rounding issues), neither Alice nor Bob's payout should change regardless of whether Alice's votes come from a single account or from multiple accounts (e.g. Alice1 and Alice2).
- The proportionality principle PP. Upvoters' curation rewards change proportionally for subsequent upvotes. A reward system that uses the secondary pool algorithm satisfies the proportionality principle.

- The ultimate indifference principle UIP. The property has the CIV property; the base reward obtained for an infinitesimal number of votes is the same for all posts regardless of upvote level.
- The DWA principle. The weight distributed per vote decreases as total votes increases.

Derivation of algorithm from principles

The micro-dynamics are driven by incoming upvotes r. The reward curve $V(r)$ represents the secondary pool's assets (primary claims); let $W(r)$ represent the secondary pool's liabilities (weight / secondary claims).

The exchange rate of weight to primary claims is $\sigma(r) = \frac{W(r)}{V(r)}$. The value of $\sigma(r)$ is the weight that represents a single primary claim when r votes have already occurred. Likewise, define $\rho(r)$ to be the exchange rate of votes to weight. In other words, $\rho(r)$ represents the votes necessary to obtain a single unit of weight (i.e. a single secondary claim) when r votes have already occurred.

An expression for $\rho(r)$ may not be immediately obvious, so let's derive it. The product of the two rates, $\sigma(r)\rho(r)$, is the number of votes which will obtain a single primary claim. But by the UIP, the votes to obtain a single primary claim must be equal to some global constant; let us call this global constant A. So set $\sigma(r)\rho(r) = A$ and substitute for $\sigma(r)$ to obtain $\rho(r) = A\frac{V(r)}{W(r)}$.

Since $\rho(r)$ represents the votes necessary to obtain a single unit of weight, it follows that $1/\rho(r)$ is the amount of weight issued for a single unit of votes. In other words, $dW/dr = 1/\rho(r)$. When we substitue for $\rho(r)$, this turns out to be a separable differential equation, which can be solved as follows:

$$\begin{aligned}
\frac{dW}{dr} &= \frac{1}{\rho(r)} = A\frac{W(r)}{V(r)} \\
\Rightarrow \frac{dW}{W(r)} &= A\frac{dr}{V(r)} \\
\Rightarrow \int \frac{dW}{W(r)} &= A\int \frac{dr}{V(r)} \\
\Rightarrow \log(W(r)) &= A\int \frac{dr}{V(r)} \\
\Rightarrow W(r) &= \exp\left(A\int \frac{dr}{V(r)}\right)
\end{aligned}$$

Defining the new curve

The new curve in v0.17 is defined as follows:

$$V(r) = \frac{r^2 + 2\alpha r}{\beta r + \gamma}$$

To get a sense for the shape of this curve, let's differentiate it:

$$\begin{aligned} V'(r) &= \frac{(2r + 2\alpha)(\beta r + \gamma) - (r^2 + 2\alpha r)\beta}{(\beta r + \gamma)^2} \\ &= \frac{2\beta r^2 + 2\alpha\beta r + 2\gamma r + 2\alpha\gamma - \beta r^2 - 2\alpha\beta r}{(\beta r + \gamma)^2} \\ &= \frac{\beta r^2 + 2\gamma r + 2\alpha\gamma}{(\beta r + \gamma)^2} \end{aligned}$$

Let's consider the slope at the origin, $m_0 = V'(0)$, and the slope at infinity, $m_\infty = \lim_{r\to\infty} V'(r)$. The ratio of these slopes, $\mu = \frac{m_\infty}{m_0}$, defines the maximum flux ratio μ:

$$\begin{aligned} m_0 &= \frac{2\alpha}{\gamma} \\ m_\infty &= \frac{1}{\beta} \\ \mu &= \frac{\gamma}{2\alpha\beta} \end{aligned}$$

Solution with new curve

Use the following formulas from Wikipedia's table of integrals:

$$\begin{aligned} \int \frac{1}{ax + b}\, dx &= \frac{1}{a} \ln|ax + b| + C \\ \int \frac{1}{x(ax + b)}\, dx &= -\frac{1}{b} \ln\left|\frac{ax + b}{x}\right| + C \end{aligned}$$

We then obtain:

$$
\begin{aligned}
V(r) &= \frac{r^2 + 2\alpha r}{\beta r + \gamma} \\
\Rightarrow \int \frac{dr}{V(r)} &= \int \frac{\beta r + \gamma}{r^2 + 2\alpha r} dr \\
&= \beta \int \frac{1}{r + 2\alpha} dr + \gamma \int \frac{1}{r^2 + 2\alpha r} dr \\
&= \beta \ln(r + 2\alpha) - \frac{\gamma}{2\alpha} \ln\left(\frac{r + 2\alpha}{r}\right) + C_1
\end{aligned}
$$

Call this last expression F. As a sanity check, let us differentiate F and make sure F' it matches the integrand:

$$
\begin{aligned}
F(r) &= \beta \ln(r + 2\alpha) - \frac{\gamma}{2\alpha} \ln\left(\frac{r + 2\alpha}{r}\right) + C_1 \\
&= \beta \ln(r + 2\alpha) - \frac{\gamma}{2\alpha} \left(\ln(r + 2\alpha) - \ln(r)\right) + C_1 \\
&= \beta \ln(r + 2\alpha) - \frac{\gamma}{2\alpha} \ln(r + 2\alpha) + \frac{\gamma}{2\alpha} \ln(r) + C_1 \\
\Rightarrow F'(r) &= \frac{\beta}{r + 2\alpha} - \left(\frac{\gamma}{2\alpha}\right)\left(\frac{1}{r + 2\alpha}\right) + \left(\frac{\gamma}{2\alpha}\right)\left(\frac{1}{r}\right) + C_1 \\
&= \frac{\beta r}{r^2 + 2\alpha r} - \frac{\left(\frac{\gamma}{2\alpha}\right) r}{r^2 + 2\alpha r} + \frac{\left(\frac{\gamma}{2\alpha}\right)(r + 2\alpha)}{r^2 + 2\alpha r} \\
&= \frac{\beta r - \left(\frac{\gamma}{2\alpha}\right) r + \left(\frac{\gamma}{2\alpha}\right) r + \left(\frac{\gamma}{2\alpha}\right) 2\alpha}{r^2 + 2\alpha r} \\
&= \frac{\beta r + \gamma}{r^2 + 2\alpha r}
\end{aligned}
$$

As we can see it matches. Now let us compute $W(r)$ and simplify the expression by naming the constant exponents $p = A/m_0 = A\frac{\gamma}{2\alpha}$ and $q = A\beta$:

$$
\begin{aligned}
W(r) = \exp\left(A \int \frac{dr}{V(r)}\right) &= C_2 \left(\frac{r}{r + 2\alpha}\right)^{\frac{A\gamma}{2\alpha}} (r + 2\alpha)^{A\beta} \\
&= C_2 \left(\frac{r}{r + 2\alpha}\right)^{p} (r + 2\alpha)^{q}
\end{aligned}
$$

The headwind term

It is tempting to choose A, β so that the exponents p, q are integers. Indeed, this approach works in the pre-HF17 quadratic reward curve, which can be regarded as the special case $\beta = 0, \gamma = 1$. The pre-HF17 code has hardcoded $A = 2\alpha$.

To gain some insight into the exponents' impact on the shape of the curve, consider the maximum flux ratio μ:

$$\mu = \frac{m_\infty}{m_0} = \frac{\gamma}{2\alpha\beta} = p/q$$

This computation makes it clear that any choice of parameters with a large ratio of slopes will require correspondingly large values of p, q. Strictly speaking it is possible to use such p, q, but the practical aspects are less than ideal: We'd be placing slow large-integer arithmetic or software floating-point in a critical, non-parallelizable code path, and using high-degree polynomials that are likely ill-behaved.

Instead we opt for $p = 1, q = \frac{1}{\mu}$ and then approximate the second term as unity. Effectively this reduces the weight function $W(r)$ by a multiplicative factor $h(r) = \left(\frac{1}{r+2\alpha}\right)^q$ which we call the *headwind term*. We may effectively fold $h(0)$ into the constant C_2; only the relative headwind term $H(r) = h(r)/h(0)$ is relevatnt to our design perspective. To simplify expression we will re-parameterize into $x = r/2\alpha$ and obtain:

$$\begin{aligned} H(x) &= \left(\frac{1}{2\alpha x + 2\alpha}\right)^q / \left(\frac{1}{2\alpha}\right)^q \\ &= \left(\frac{1}{x+1}\right)^q \end{aligned}$$

It is informative to find values of x for which $H(x)$ corresponds to power-of-two fractions $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \ldots$:

$$\begin{aligned} H(x) = \left(\frac{1}{x+1}\right)^q &= \left(\frac{1}{2}\right)^z \\ \Leftrightarrow q \log\left(\frac{1}{x+1}\right) &= z \log\left(\frac{1}{2}\right) \\ \Leftrightarrow -q \log(x+1) &= -z \log(2) \\ \Leftrightarrow \log(x+1) &= \log(2)\left(\frac{z}{q}\right) \\ \Leftrightarrow x+1 &= 2^{z/q} \\ \Leftrightarrow x &= 2^{z/q} - 1 = 2^{\mu z} - 1 \end{aligned}$$

As we can see, the headwind term falls by half at exponentially increasing intervals.

The scale parameter

Next we will turn our attention to considering $\phi(r) = V'(r)/V'(0)$:

$$\begin{aligned} V'(r) &= \frac{\beta r^2 + 2\gamma r + 2\alpha\gamma}{(\beta r + \gamma)^2} \\ \Rightarrow \phi(r) = V'(r)/V'(0) &= \left(\frac{\beta r^2 + 2\gamma r + 2\alpha\gamma}{(\beta r + \gamma)^2}\right) / \left(\frac{2\alpha\gamma}{\gamma^2}\right) \\ &= \left(\frac{\beta r^2 + 2\gamma r + 2\alpha\gamma}{(\beta r + \gamma)^2}\right) \left(\frac{\gamma}{2\alpha}\right) \end{aligned}$$

This equation is somewhat unwieldy, so let us introduce the simplifying approximations $r, \beta r \ll \gamma$ and proceed:

$$\begin{aligned} \phi(r) &= \left(\frac{\beta r(r + 2\gamma) + 2\alpha\gamma}{(\beta r + \gamma)^2}\right) \left(\frac{\gamma}{2\alpha}\right) \\ &\approx \left(\frac{2\beta r\gamma + 2\alpha\gamma}{\gamma^2}\right) \left(\frac{\gamma}{2\alpha}\right) \\ &= (\beta r + \alpha) \left(\frac{1}{\alpha}\right) \\ &= 1 + \frac{\beta r}{\alpha} \end{aligned}$$

If we define a point s on the curve such that $\phi(s) = 2$, analogously with the quadratic case, we find that $s \approx \frac{\alpha}{\beta}$.

Final parameter choices

To evaluate $V(r), W(r)$ we need to define four parameters: A, α, β, γ. Instead of defining these parameters directly, we instead set $p = 1, \beta = 1$ and choose μ, s; the other parameters are constrained as follows:

$$\begin{aligned} p &= 1 = \frac{A\gamma}{2\alpha} \\ \mu &= \frac{\gamma}{2\alpha\beta} \\ s &= \frac{\alpha}{\beta} \end{aligned}$$

These constraints can be solved to fix the curve parameters as functions of the chosen parameters:

$$\begin{aligned} \alpha &= \beta s \\ \gamma &= 2\alpha\beta\mu \\ A &= \frac{2\alpha}{\gamma} = m_0 \end{aligned}$$

To summarize the effects of the free parameters:

- s specifies where "takeoff" takes place and bonus rewards become greater than base rewards.
- β affects how fast "saturation" occurs and the rate of bonus reward increase slows. In particular when $r = \frac{\gamma}{\beta} = 2\alpha\mu = 2\beta s\mu$, rewards accumulate half as fast as with the old quadratic reward curve.
- μ is the ratio of maximum possible bonus + base rewards to base rewards.