An Open Standard Algorithm for Rarity Rankings

Rarity Punks
raritypunks.io

Abstract. An open standard for rarity rankings would allow collectors to evaluate the rarity rank of a collectible within a collection without the need for a centralized rarity rankings authority. Since rarity rankings can explain a large degree of the variation in market prices for assets within a collection, relying upon a centralized authority to produce rarity rankings is open to corruption by insider manipulation. We propose a reproducible and robust standard algorithm to produce rarity ranks, with the option for creators to define weightings and tiers to align rarity rankings with the creative vision of the project. This prevents insiders from nefariously manipulating rarity rankings to suit their own purposes, and also allows collectors to know what they are trading, no later than insiders could.

1. Introduction

One of the most important drivers of value in the trading of collectibles is rarity. For instance, the highest sale of one of the earliest non-fungible token (NFT) collectibles, CryptoPunks, was more than 40 times higher than a recent sale [1]. This variation can be largely explained by rarity, as evidenced by the more expensive CryptoPunk being one in only nine Alien punks out of a collection of 10,000.

Trusted central rarity ranking authorities have emerged in order to generate rarity rankings for NFT collections. While this system works well enough for most collections, it suffers from the potential for rarity ranking manipulation. As demonstrated with the earlier example involving CryptoPunks, there is a very large financial incentive to exploit rarity rankings in one's favor. One example of this was publicized in the MekaVerse launch, which was one of the most anticipated NFT collections in 2021. Heavy public scrutiny exposed a number of issues [2][3] that enabled insiders to trade on the rarity of tokens before the public could. Furthermore, when the rarity rankings were published by a leading central rarity ranking authority, trait counting had been disabled [4] despite the fact that this is a commonly used feature in the rarity rankings for most collections (in this particular case, this had a negligible effect on rankings). However, a number of collections still choose to adopt a different mathematical approach to rarity rankings than the vast majority of collections. This is only known days, or sometimes over a week, after trading on revealed metadata has begun. Due to the centralized nature of central rarity ranking authorities, who are often being paid by the collection, it is often unclear and opaque why certain rarity ranking decisions were made. Although it is hard to definitively prove that wilful manipulation of rankings has occurred, it should not be ruled out since there is a very compelling financial motive and bad actors have already been observed to exploit insider trading on rarity to their advantage.

What is needed is an open and decentralized standard to calculate rarity rankings, so that anyone can know the rarity of what they are trading at the exact same time as insiders can. This removes the power of central rarity ranking authorities to set up different rules for different collections, which currently allows insiders to manipulate the rules to produce ranks to benefit themselves at the expense of the public. It is also worth noting that there are other important vulnerabilities that allow the rarity to be exploited by insiders that are not in the scope of this document (such as metadata changes and leaks, and the use of IPFS and verifiable random functions). Ultimately, we believe that an open standard for rarity rankings will prevent insiders from nefariously manipulating rarity rankings to suit their own purposes, and lets collectors know the rarity of what they are trading, no later than insiders could.

2. Attributes Metadata

Within the metadata for each token in a collection, there is a list of attributes that defines a series of traits with a corresponding value. For example, a particular token may have a *Hat* trait with the value *Baseball Cap*. This is typically defined in a token's metadata in JSON format as follows:

```
"attributes": [
    {
        "trait_type": "Hat",
        "value": "Baseball Cap"
    },
    ...
]
```

For a given trait, there may be many possible values across the entire collection. For example, for the *Hat* trait, the corresponding values seen across the entire collection may include *Baseball Cap*, *Fedora*, and *Straw Hat*. To calculate the rarity ranking for every token in a given collection, the attributes metadata is assumed to be known for every token in the collection.

In cases where a collection is only partially revealed, rarity rankings can still be calculated on the revealed set of tokens, but will only be valid for that set of tokens. Once metadata becomes available for additional tokens, the rarity rankings will need to be recalculated from scratch. This can change the rankings of previously revealed tokens, as the newly revealed tokens can be ranked better than an already revealed token, thereby displacing its original ranking.

3. Frequency Counting

The starting point of rarity calculations begins with counting the frequency with which a value (of a given trait) appears in the collection. All else being equal, the lower the frequency of occurrence, the more rare it is. Therefore, a base score could be proposed, as follows:

s_B = base score of a value of a given trait
n = the number of times a value in a given trait appears in the collection (frequency)
$s_B = 1/n$

So if there are 20 *Baseball Cap*s and 10 *Fedora*s of the *Hat* trait, a token with a *Fedora* would score double the score of a token with a *Baseball Cap*. Using this process, a score can be determined for every value in the token's attributes metadata.

4. Total Score

Each metadata value in the token can be given a component score from the base score, as follows:

s_i = score assigned to value i of a given trait j
$s_{Bi} = 1/n_{i,j}$ = base score of value i of a given trait j
k_j = constant for a given trait j
$s_i = k_j \times s_{Bi}$

Either a geometric combination or an arithmetic combination could be used to derive a token's overall rarity score from the component scores of each of its metadata values, as follows:

$$S_g = (s_1 \times s_2 \times s_3 \times \cdots)$$
$$S_a = (s_1 + s_2 + s_3 + \cdots)$$

To illustrate the suitability of each approach, suppose there are two tokens as follows:

Token A: $s_1 = 1$, $s_2 = 1/1000$ $\Rightarrow S_g = 1/1000 = 0.0010, S_a = 1.001$
Token B: $s_1 = 1/30$, $s_2 = 1/30$ $\Rightarrow S_g = 1/900 \approx 0.0011, \; S_a = 1/15 \approx 0.067$

In this example, Token A represents a token with a very rare component and a common component, while Token B represents a token with two uncommon components. Token B performs better than Token A under the geometric approach, while Token A performs significantly better than Token B under an arithmetic approach. According to the geometric approach, the coincidence of two uncommon components is less probable than a very rare component coinciding with a common component. Under this approach, the rare value in Token 1 is being devalued by the presence of the common component. On the other hand, the arithmetic approach fully incorporates the value of the rare component, and is barely impacted by the relatively insignificant common component.

In practice, rarity is better described by the arithmetic combination, since a common component does not tend to detract from the worth of a collectible that already has a very rare component. Furthermore, two uncommon components are not equivalent to one very rare component, as the worth of a collectible tends to resemble a hyperbolic-like relationship with respect to a component's rarity. Lastly, the arithmetic approach is already the widely used and dominant methodology currently adopted by rarity ranking sites [5]. For these reasons, it is proposed that a token's overall rarity score should be constructed using an arithmetic combination of the component scores of a token's metadata values, as follows:

$$S = s_1 + s_2 + s_3 + \cdots$$

5. Trait Equalization

Computing an overall rarity score as the sum of each component's score can result in a bias that favors tokens that have more components than other tokens, since a component score is always greater than zero. In order to correct against this bias, a component score should be added in to evaluate the frequency of a missing value for a trait. This component score can be calculated in the same way as any other value is calculated. For example, if there are 10 tokens with 8 tokens with the value *Sword* as the *Weapon* trait and 2 remaining tokens with no *Weapon* trait, the tokens with no *Weapon* trait should have a component score that is four times greater than a token with the value *Sword* as the *Weapon* trait. The higher score for an empty *Weapon* trait reflects the rarity of not having a *Weapon* trait.

Sometimes multiple values are assigned to the same trait. For example, there could be a token with multiple values under the *Accessory* trait, such as a *Ribbon* value and a *Ring* value, as follows:

```
"attributes": [
    {
        "trait_type": "Accessory",
        "value": "Ribbon"
    },
    {
        "trait_type": "Accessory",
        "value": "Ring"
    },
    ...
]
```

In order to maintain component equality, tokens with fewer values of a given trait than any other token in the collection should add in component scores to compensate for every missing value. In order to score this component, the frequency of empty occurrences can be calculated by:

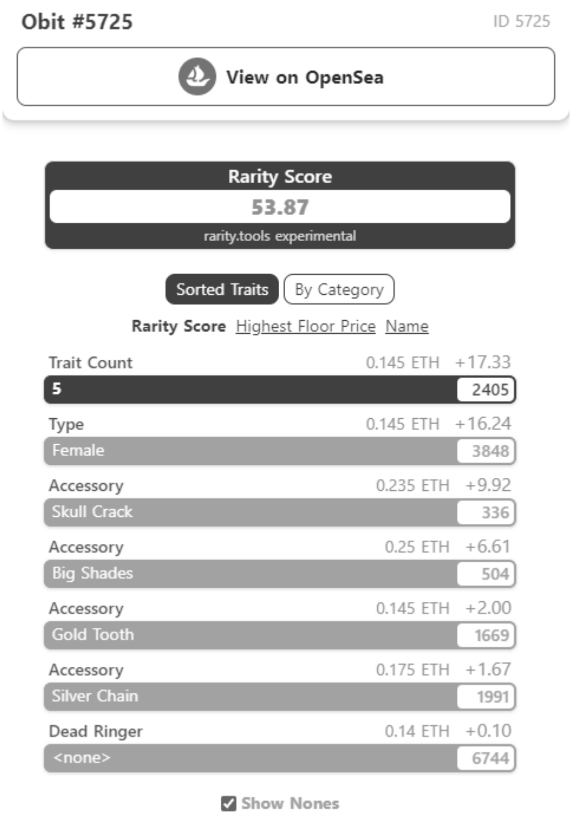
n_0 = the frequency of empty occurrences for a given trait in the collection
n_i = the number of times that value i of a given trait appears in the collection
m = maximum number of times that a given trait appears in any token in the collection
N = the total number of tokens in the collection
$n_0 = Nm - \Sigma n_i$

This formula also works with the simpler $m = 1$ case initially described, as $n_0 = (10)(1) - (8) = 2$. Using a more complex example, if there are 10,000 tokens with up to $m = 4$ *Accessory* traits defined for a single token, and a total of 31,200 values of the *Accessory* trait defined across the collection, then the frequency of empty occurrences is:

$n_0 = (10{,}000)(4) - (31{,}200) = 8{,}800$

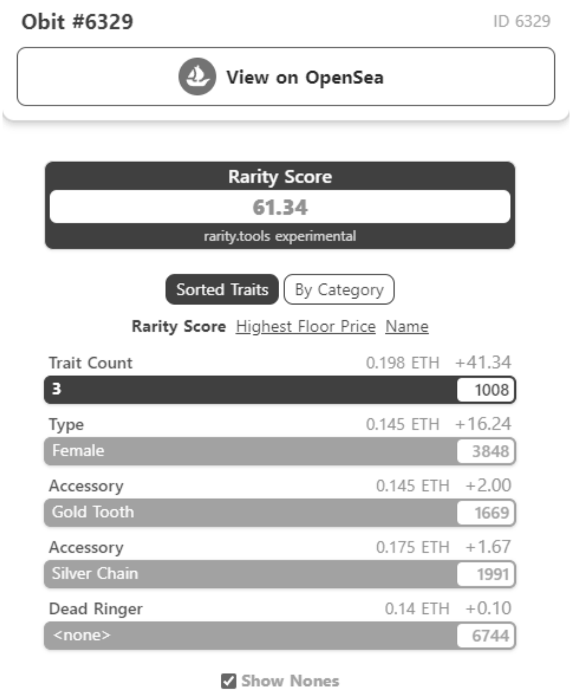
In this example, the average number of values under the *Accessory* trait is 31,200/10,000 = 3.12.

At the time of writing, we are not aware of any examples of rarity ranking authorities that score empty traits beyond the $m = 1$ case correctly. Thus, these rarity rankings overvalue tokens that have multiple assignments of a trait, as these tokens have extra score components than other tokens. An example of this error, taken from a widely used central rarity ranking authority, Rarity.Tools, is shown for two different Obits tokens with a different count of *Accessory* scores below [6][7].


Obit #5725 ID 5725
View on OpenSea
Rarity Score
53.87
rarity.tools experimental
Sorted Traits | By Category
Rarity Score Highest Floor Price Name

```
Trait Count        0.145 ETH  +17.33
5                              2405
Type               0.145 ETH  +16.24
Female                         3848
Accessory          0.235 ETH   +9.92
Skull Crack                     336
Accessory          0.25 ETH    +6.61
Big Shades                      504
Accessory          0.145 ETH   +2.00
Gold Tooth                     1669
Accessory          0.175 ETH   +1.67
Silver Chain                   1991
Dead Ringer        0.14 ETH    +0.10
<none>                         6744
```

Show Nones


Obit #6329 ID 6329
View on OpenSea
Rarity Score
61.34
rarity.tools experimental
Sorted Traits | By Category
Rarity Score Highest Floor Price Name

```
Trait Count        0.198 ETH  +41.34
3                              1008
Type               0.145 ETH  +16.24
Female                         3848
Accessory          0.145 ETH   +2.00
Gold Tooth                     1669
Accessory          0.175 ETH   +1.67
Silver Chain                   1991
Dead Ringer        0.14 ETH    +0.10
<none>                         6744
```

Show Nones

Obit #5725 has 4 component scores for the *Accessory* trait, while Obit #6329 only has 2 component scores for the *Accessory* trait. This biases the score in favor of Obit #5725 as Obit #6329 is not being credited for 2 missing components where the *Accessory* trait is empty.

6. Trait Normalization

Until now, we have not defined the constant k_j used to compute the component score of a given trait j:

$$s_i = k_j \times s_{Bi}$$

Although some rarity ranking sites set $k_j = 1$ for all j, the majority of rarity rankings on Rarity.Tools incorporate trait normalization. Trait normalization corrects a bias that occurs due to varying average frequencies for the values of different traits in the collection. For example, a collection with 10,000 tokens and 5 different values for the *Background* trait and 20 different values for the *Weapon* trait will have a lower average frequency for *Weapon* values (10,000/20 = 500) than *Background* values (10,000/5 = 2,000). Without trait normalization, the base score of *Background* values is simply added to the base score of *Weapon* values, and on average the base score of the *Weapon* value will be 4 times greater than the base score of the *Background* value. Trait normalization corrects this bias by comparing values against the average value for that trait. Thus, the constant k_j could be defined as follows:

u_j = unique number of values of a given trait j (including a missing value as a unique value)
N = the total number of tokens in the collection
$k_j = N/u_j$

This multiplies the base score by the average frequency of a trait. This compensates for traits where its values have a higher average frequency, which would otherwise appear less rare due to the higher frequency of occurrence.

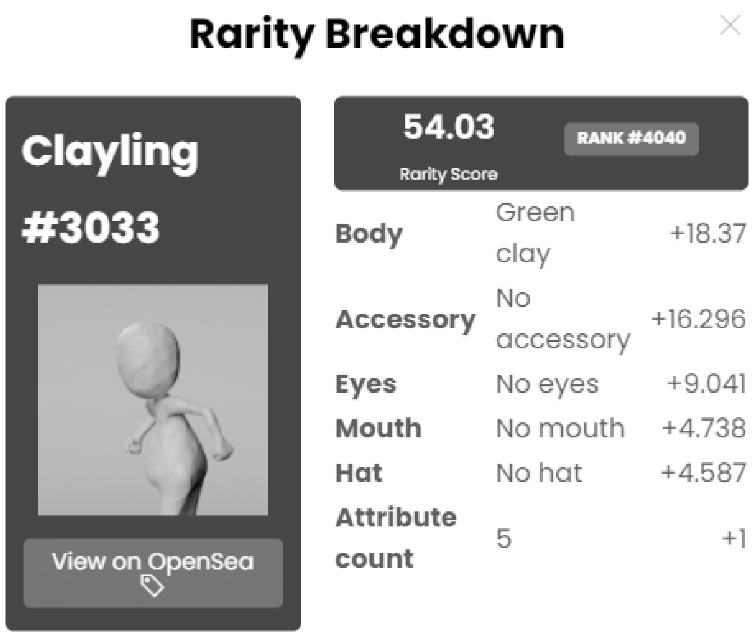
7. Trait Counting

Another common feature of rarity ranking algorithms is trait counting, which dates back to one of the earliest NFT collections, CryptoPunks [8]. In the CryptoPunks collection, there is only one CryptoPunk with 7 traits and eight CryptoPunks with 0 traits. The most common number of traits is 3. Rarity ranking sites rank the CryptoPunk with 7 traits in first place of the entire collection, and the eight CryptoPunks with 0 traits in the top 1% of the collection.

This can be taken into account by including an extra component score for the trait count of a token into the overall rarity score, added alongside any other component and normalized, as follows:

n_c = the number of times a trait count of c appears in the collection (frequency)
$s_{Bc} = 1/n_c$ = base score of a trait count of c
u_c = unique number of possible trait counts (e.g. 0, 1, 2, 3, 4, 5, 6, or 7 means $u_c = 8$)
N = the total number of tokens in the collection
$k_c = N/u_c$
$s_c = k_c \times s_{Bc}$
$S = (s_1 + s_2 + s_3 + \cdots) + s_c$

Improper implementation of trait counting is a common mistake amongst some rarity ranking sites. In the following example, NiftyRiver mistakenly places a 1 of 1 Clayling in rank #4040 [9], which is the lowest possible rank in Claylings collection of 4040.



Rarity Breakdown

Clayling #3033

View on OpenSea

54.03 Rarity Score RANK #4040

Body	Green clay	+18.37
Accessory	No accessory	+16.296
Eyes	No eyes	+9.041
Mouth	No mouth	+4.738
Hat	No hat	+4.587
Attribute count	5	+1

In the above example, the values *No accessory*, *No eyes*, *No mouth*, and *No hat* are mistakenly counted in the trait count. If these values were treated as empty traits, Clayling #3033 would have a trait count of 1 instead of 5, which would make it a 1 of 1 in the entire collection.

To avoid this issue, we propose that developers avoid defining empty traits as a string, and instead simply leave the trait out of a token's attributes metadata. Alternatively, developers could list traits by adding rarity metadata, duplicated across each token, to instruct rarity ranking sites to ignore certain values for given traits for the purposes of trait counting, as shown below:

```
"attributes": [
    ...
],
"rarity": {
    "count_as_empty": [
        {
            "trait_type": "Accessory",
            "value": "No accessory"
        },
        {
            "trait_type": "Eyes",
            "value": "No eyes"
        },
        {
            "trait_type": "Mouth",
            "value": "No mouth"
        },
        {
            "trait_type": "Hat",
            "value": "No hat"
        }
    ]
}
```

8. Trait Weighting and Tiers

A potential disadvantage of using a predefined standard algorithm for rarity rankings is that it may be difficult for creators to align rarity rankings with the creative vision of their project. A simple solution to this problem is to define a standard schema of weights and tiers that can be included within the token metadata. With an open standard algorithm, creators can simulate the rarity rankings of their creations in advance of any metadata reveal, instead of waiting until after the collection is revealed to work with a central rarity rankings authority. More importantly, this allows collectors to know the rarity of what they are trading no later than insiders would.

Weights allow creators to apply a greater emphasis on specific traits. For instance, if the *Species* trait is particularly important to the collection, and the less common values are not being appropriately valued by the standard algorithm, a weighting can be applied to inflate the value of any component score relating to the *Species* trait. This can be implemented via the constant k_j used to compute the component score of a given trait j:

$$s_i = k_j \times s_{Bi}$$

Earlier, k_j was defined based on trait normalization. However, an additional parameter can be added in to also factor in a weighting factor w_j, as follows:

w_j = weighting multiplier of a given trait j (default setting is 1)
u_j = unique number of values of a given trait j (including a missing value as a unique value)
N = the total number of tokens in the collection
$k_j = w_j \times N/u_j$

This weighting can be used to amplify or diminish the effect of any given trait. Values of w_j greater than 1 have an amplification effect, while values of w_j less than 1 have a diminutive effect. Weights can be combined across a range of traits in order for creators to develop rarity rankings to align with their creative intent.

To declare weights, we propose that developers specify weightings in rarity metadata, duplicated across each token, to instruct rarity ranking sites to apply the desired weightings, as shown below:

```
"attributes": [
    ...
],
"rarity": {
    "weights": {
        "Species": 3,
        "__trait_count": 1
    }
}
```

For illustrative purposes, `__trait_count` (with two leading underscores) is included to demonstrate how to set the weighting for the trait count score. Since the default weight assumed is 1, it would not normally need to be specified as above.

Tiers allow creators to segment their collection into two or more separate subcollections. For instance, the CyberKongz collection began with 1,000 Genesis tokens alongside a growing number of Baby tokens [10]. These are defined in the metadata under the *Type* trait, but Genesis tokens are much more valuable than Baby tokens due to certain privileges afforded to Genesis tokens. Rather than set a large weighting on *Type*, which would only be a feasible option if there were fewer Genesis tokens than Baby tokens, it is simpler to treat the collections separately for the purposes of rarity rankings. Under a tiering system, the rarity ranking algorithm is simply applied on each tier as if it were its own collection. Thus, in the CyberKongz example, there would be a top-ranked Genesis token, as well as a top-ranked Baby token.

To instruct rarity ranking sites to treat different values of a given trait as a different tier, we propose that developers specify a tier field in the rarity metadata, as shown below:

```
"attributes": [
    ...
],
"rarity": {
    "weights": {
        "Species": 3
    },
    "tier": "Type"
}
```

Only one trait can be set as a tier, as this would create a high degree of permutational complexity.

9. Rarity Rankings

In summary, the overall rarity score can be assembled for each token as follows:

$S = (s_1 + s_2 + s_3 + \cdots) + s_c$
$s_i = w_j \times N/u_j \times 1/n_{i,j}$ = score of the *i*th metadata value, of trait *j*, of a given token
$s_c = w_c \times N/u_c \times 1/n_c$ = score of the trait count of a given token
w_j = weighting multiplier of a given trait *j* (default setting is 1)
w_c = weighting multiplier of trait count (default setting is 1)
N = the total number of tokens in the collection
u_j = unique number of values of a given trait *j* (including a missing value as a unique value)
u_c = unique number of possible trait counts
$n_{i,j}$ = the number of times value *i* in a given trait *j* appears in the collection (frequency)
n_c = the number of times a trait count of *c* appears in the collection (frequency)

For each tier, the overall rarity scores for each token are ranked in descending order to determine the rank of that token, with tied scores treated as equal ranks.

10. Conclusion

Relying upon centralized actors to choose a rarity algorithm allows corruption and rigging to take place. Given the importance of rarity in determining the value of an NFT, insiders can manipulate rarity rankings to best suit themselves and their associates, at the expense of others. To prevent this, we proposed an open standard for rarity ranking that would disallow a central rarity ranking authority from being able to change the rules at the request of their paying customers. An open standard algorithm allows anyone to reproduce rarity rankings, no later than insiders could. Incorporated into the open standard algorithm is trait counting and trait normalization, which most collections have already adopted. Creators can still align rarity rankings with their creative intent by using weightings and tiers, which can be published with the token metadata. While we understand that historical collections already have entrenched trade patterns around legacy rarity ranking algorithms, we suggest that all future collections adopt an open and decentralized standard rarity ranking algorithm, alongside the open and decentralized networking principles that underpin the existence of cryptocurrencies and NFTs.

References

[1] Larva Labs, “CryptoPunks: Top Sales”, https://larvalabs.com/cryptopunks/topsales, October 2021

[2] OKHotshot.eth, “Finished on-chain analysis of the MekaVerse NFT drop…”, https://twitter.com/NFTherder/status/1448750746967683075, October 2021

[3] Beanie.eth, An account using insider information to bid on rares before public knowledge was known, https://twitter.com/beaniemaxi/status/1448526431198318593, October 2021

[4] Rarity.Tools, “MekaVerse Ranked by Rarity”, https://rarity.tools/mekaverse, October 2021

[5] Rarity.Tools, “Ranking Rarity”, https://raritytools.medium.com/ranking-rarity-understanding-rarity-calculation-methods-86ceaeb9b98c, May 2021

[6] Rarity.Tools, “Obit #5725”, https://rarity.tools/obitsofficial/view/5725, October 2021

[7] Rarity.Tools, “Obit #6329”, https://rarity.tools/obitsofficial/view/6329, October 2021

[8] Larva Labs, “CryptoPunks: All Attributes”, https://www.larvalabs.com/cryptopunks/attributes, October 2021

[9] NiftyRiver, “Clayling #3033”, https://www.niftyriver.io/rarity/0x8630cdeaa26d042f0f9242ca30229b425e7f243f?q=&token_id=3033, October 2021

[10] CyberKongz, “About CyberKongz”, https://www.cyberkongz.com/about, October 2021