In this experiment, we wanted to rank developers, identified by their GitHub accounts, based on their proximity to OP Stack repos. We trialed a unique approach with a custom implementation of the EigenTrust algorithm.

Special thanks to Carl Cervone (Open Source Observer), Emily Furlong (Optimism Foundation) and Eugene Kim (OpenRank), Jenya Nepoimannykh (OpenRank).

The Foundational Method: Capturing the Web of Trust via Recursive Trust Graph

Given a community of peers, their interactions with each other form a directed graph. Here, the peers become nodes within the graph and their interactions are the graph edges. These asymmetric edges can be viewed as trust arcs from one peer to the other. Additionally, the weight of each edge signals the degree of trust imparted by one peer onto another.

The EigenTrust algorithm can then be run on such graphs to measure the impact of recursive trust propagation, leading to a global trust value for each peer or node in the network. Thus, every node gets assigned a reputation score. In a bipartite graph, there are two types of nodes, for example, GitHub repos as well as GitHub users as givers and receivers of trust forms a bipartite graph. EigenTrust not only works well with peer-to-peer data, but it can also be used for bipartite graphs, the implementation of which is achieved using the Hubs and Authorities algorithm. In a bipartite graph, there are two types of nodes, for example, GitHub repos as well as GitHub users as givers and receivers of trust forms a bipartite graph.

The Goal of the Exercise and the Choice of Algorithm

Goal

To rank developers, identified by their GitHub accounts, based on their proximity to OP Stack repos.

Approach

OpenRank enables verifiable compute for a large class of reputation algorithms, **in particular those that (a) operate on a graph, (b) are iterative in nature and (c) have a tendency toward convergence. In the following sections, we give a brief overview of some of these algorithms.

For our Developer ranking use case, we obtained a graph of relevant GitHub event data from Open Source Observer, covering more than 2,000 organizations and over 30,000 code repositories. We established two kinds of peers: GitHub users and repositories. We then used a variation of EigenTrust, the Hubs and Authorities algorithm, which computes rankings on an asymmetric bipartite trust graph.

Below are the two different trust edges used in the graph:

- User-to-repository trust
- this signals the user’s interest in the repository through actions such as stars, forks and issue/PR submissions.
- Repository-to-user trust
- this signals the credit extended by the repository to the user for their contribution. Actions such as PRs merged, direct code commits received, and other maintenance actions.

The reason for using these two different and asymmetric trust arcs between users and GitHub repositories is to ensure accuracy and sybil resistance in the rankings. For trust-critical use cases, shared interest signal is often not necessarily a good measure of trust. For example, sybil actors can create thousands of trust arcs directed to a malicious or irrelevant repository. We solve this in two ways, choose a set of somewhat trustworthy/useful seed peers (GitHub repositories); and using a trust arc from a repository to a user based on a relevant credit signal such as merging a PR.

Once the compute is run, the resulting global trust can be used for ranking both users and GitHub repositories.

Compute Model Assumptions and Weights

Trust Signals

In this model, we use developers (GitHub username) and OP Stack repositories as nodes in the trust graph. We establish trust arcs between users and repositories and vice versa in the following manner:

User-to-repository actions (interest signals) include:

- Starring the repository

- Forking the repository
- Opening an issue
- Opening a pull request
- Directly committing to the repository

Repository-to-user actions (credit signals) include:

- Having a pull request merged into the repository (approximated by opening of a pull request as of this exercise)
- Directly committing to the repository
- Repository maintenance, such as merging a PR, resolving an issue, etc.

Choice of Seed Peers and Alpha Value

Seed peers are core OP Stack repo, weighted by their relative importance.

- Core repos with higher importance (weight =2)

‘testinprod-op-erigon’

‘a16z/magi’,

‘ethereum-optimism/optimism’,

‘ethereum-optimism/op-geth’,

‘ethereum-optimism/asterisc’,

‘ethereum-optimism/kona’,

‘ethereum-optimism/design-docs’,

‘ethereum-optimism/specs’

- Other core repos with importance (weight=1)

‘ethereum-optimism/superchain-ops’,

‘ethereum-optimism/op-analytics’

Alpha (α) Value

α is a parameter in EigenTrust algorithm that determines the significance of seed peers influence that shapes the reputation score of every other peers in the entire network. Alpha value is normally chosen between 0 and 0.5 and the higher the alpha the stronger the influence of seed peers, meaning the trust arc from seed peers to other peers will matter more for a peer’s relative reputation.

A weak alpha value (α=0.2) is chosen in this exercise, intending to discover more repos related to OP stack, and attribute contribution back to developers. Relatively less influential seed peers could let the other peers express more reputation credit to other repos and peers, therefore suits the purpose of the exercise. To read more about alpha value, check out the EigenTrust documentation here.

Reasoning of the Weights Choices for Trust Arcs

The weight ratio in the same class of actions (user-to-repo, or repo-to-user) matters.

User-to-repo weights signals the seriousness of interest as well as the effort that would go into the action. Least to most significant: Forking < (committing directly, starring) < (opening issue, merging PR) < opening PR. A relative weight was assigned accordingly.

- Forking is a chore when someone wants to take a look at the repo and try something out quickly. (weight=1)
- Starring is a long-term interest signal to get updates from the repo and also skew the feed. (weight=5)
- Opening an issue actually takes nonnegligible effort. (weight=10)
- Opening a PR takes more effort as it also has proposed changes. PR opening reflects an original work. (weight =20)
- PR merging reflects a maintenance/review action. Typically it takes more effort for someone to create a patch and

open a PR than to review it. (weight =10)

- Direct commits would normally sit between opening an issue and opening a PR; there are a lot of automated direct commits/pushes so we dial it down to the same level as opening an issue. (weight=5)

Repo-to-user weight signals the degree of credit “thank you” from the repo(’s maintainers) to contributors (users). It can be thought of credit/“thank you” from the repository to the contributors. Least to most significant: merging PR (maintainer action) < direct commits (maintainer/allow-listed contributor action) < authoring and submitting a PR (which was eventually merged). A relative weight was assigned accordingly.

Iteration on the Ranking Results

There were several rounds of iteration on the algorithm, but it should by no means be interpreted to be in its final state. The process of adjusting and improving an algorithm of this type is a continuing effort and we imagine that this algorithm still has room for improvement, but is of sufficient quality for use in this experiment.

The first iteration of the algorithm appeared to overvalue outdated repos, so we added a time decay function. We also adjusted the weight of seed repos to reflect the relative importance to the OP stack. In running the algorithm we discovered that the choice between an alpha value of 0.5 and 0.2 did not have a significant effect on the ranking.

Discussion: Future Optimizations and Sybil-resistance

This is the first iteration of using OpenRank to compute reputation within OP Stack based on data inputs from Open Source Observer, eg: Github event data. A few topics can be explored further.

Additional Signals for Increasing Model Performance and Sybil Resistance

- Using real (non-approximated) signal of having a pull request merged into the repository
- Instead of approximating “opened PR = merged PR”
- Instead of approximating “opened PR = merged PR”
- Parsing GitHub events for more concrete signs of collaboration (eg, comments on issues, co-authored pull requests) and assessing the quality of those collaboration signals (eg, a conversation taking place over multiple weeks might be higher signal than one over a shorter interval)
- Both interest signals and credit signals are captured in the same local trust matrix. The local trust may also capture user-user and repo-repo trust signals (the latter being, once again, the success attribution signals), so the graph is not necessarily bipartite. Some examples:
- Following/Starring other users
- Repository dependencies (if repo A depends on repo B, A’s success should be attributed to B)
- Following/Starring other users
- Repository dependencies (if repo A depends on repo B, A’s success should be attributed to B)

Factoring Action’s Size into the Trust Arc Weight

- Example: PR importance (e.g. documentation fix < a whole new feature)
- Subjective factor (what kind of PR is it? feature? bug fix? doc? does the project use labels for these?)
- Objective factor (change size?)
- Subjective factor (what kind of PR is it? feature? bug fix? doc? does the project use labels for these?)
- Objective factor (change size?)

Expanded Seed Trust

- Use a wider selection of OP-related repos so users’ contribution to those repos will directly increase the user’s trust standing via the seed trust
- Try and include both actors and objects in the seed. Actors-only or objects-only seed trust often yields to slow convergence.
- Associate different weights to seed repos, e.g. according to their importance

About OpenRank

OpenRank is a decentralized ranking and reputation protocol. It enables a verifiable reputation compute layer for the open web that unlocks a broad range of useful applications, including those that resist cryptographic or game-theoretic mechanisms of trust. Using graph compute algorithms like EigenTrust, it offers resilience from sybil contexts, provides scalable and context-specific compute, and enables permissionless access to compute and reputation data for any developer.

The Verifiability of OpenRank powered Reputation Scores

As a reputation protocol, the permissionless verifiability of the scores is a key property of OpenRank's proposition. The verification methods that we propose are far cheaper than verification of general purpose computations, based on the intuition that iterative convergent algorithms on a graph are expensive to compute, but inexpensive to check (typically requiring one matrix-vector multiplication to check for convergence). This makes it far more practical to use this reputation computer for these use cases than a general purpose decentralized compute network.

Contact gloria@karma3labs.com to access the full preliminary litepaper.

To run the code yourself, start by creating a Google Cloud account and subscribe to the dataset following the below guides:

docs.opensource.observer

Get Started | Open Source Observer

There are two easy ways of accessing OSO datasets: through our GraphQL API

docs.opensource.observer

Data Overview | Open Source Observer

First, you need to set up your BigQuery account. You can do this by going to the

Github Link of the Open Sourced Code

github.com

openrankprotocol/openrank-notebooks/blob/main/OpenRank + GitHub I&C for Retrofunding R5 Voter Selection.ipynb

{ "nbformat": 4, "nbformat_minor": 0, "metadata": { "colab": { "provenance": [] }, "kernelspec": { "name": "python3", "display_name": "Python 3" }, "language_info": { "name": "python" }, "widgets": { "application/vnd.jupyter.widget-state+json": { "5d29ff5c680141db91e6e36a92d0902d": { "model_module": "@jupyter-widgets/controls", "model_name": "HBoxModel", "model_module_version": "1.5.0",

This file has been truncated. show original