Web3Recommend

Decentralised Web3 social recommendations with trust and relevance balance

— MSc. Thesis —

Rohan Madhwal
Delft University of Technology
Delft, The Netherlands
R.Madhwal@student.tudelft.nl

Johan Pouwelse
Delft University of Technology
Delft, The Netherlands
J.A.Pouwelse@tudelft.nl

Abstract*—Web3Recommend is a decentralised Social Recommender System implementation which enables Web3 Platforms on Android to generate recommendations that balance trust and relevance. Generating recommendations in decentralised networks is a non-trivial problem because these networks suffer from the lack of a global perspective due to the absence of a central authority. Further, decentralised networks are prone to Sybil Attacks in which a single malicious user can generate multiple fake or "sybil" identities, allowing them to unfairly manipulate the ranking of items, thus leading to low quality, untrustworthy recommendations. The spam created by sybil identities also places a large burden on Web3 platforms due to resource wastage. Web3Recommend relies on a novel graph-based content recommendation design inspired by GraphJet, a recommendation system used in Twitter enhanced with MeritRank, a decentralised reputation scheme which provides sybil-resistance to the system. By adding MeritRank's decay parameters to the vanilla Social Recommender Systems's personalised SALSA graph algorithm, we can provide theoretical guarantees against Sybil Attacks in the generated recommendations. We leverage the efficient graphing capabilities from jGraphT in our end-to-end decentralised Android implementation. By using a real-time edge gossiping mechanism on semantic overlays in combination with compact serialization schemes, we construct global knowledge of the network on each node. Performing probabilistic Monte Carlo method based analysis of the social network structure, our system is able to rank all items available to a user in order of relevance while limiting sybil influence on the rankings. Similar to GraphJet, we focus on generating real-time recommendations by only acting on recent interactions in the social network, allowing us to cater temporally contextual recommendations while keeping a tight bound on the memory usage in resource constrained devices, allowing for a seamless user experience. As a proof-of-concept, we integrate our system with MusicDAO, an open-source Web3 music sharing platform to generate personalised, real-time recommendations. Thus, we provide the first sybil-reistant social recommender system which allows real-time recommendations that go beyond classic user based collaborative filtering. The system is also rigorously tested with extensive unit and integration tests. Further, our experiments demonstrate the trust-relevance balance of recommendations against multiple adversarial strategies in a test network generated using data from real music platforms.

I. Introduction

The recent decade has witnessed an explosion of user generated data on the Internet. According to a study from IBM called "The Big Data Problem", users generate 2.5 quintillion bytes of data on a daily basis. In fact, 90 percent of the data in the world today was created in the last two years alone. [1]

While this explosive growth in the amount of digital information available online provides a plethora of options for a diverse range of users and interests, it also results in the hindrance of timely access to items of interest and relevance since finding anything useful requires time-intensive sifting through troves of data, a majority of which is often entirely irrelevant. [2], [3] In the words of neuroscientist Daniel J. Levitin, "The information age is drowning us with an unprecedented deluge of data". [4]

The harms of information overload exceed beyond simple time wastage with studies showing that it can lead to decrease in efficiency, increased stress and even ill-health. [5]

The problem is exacerbated in social media platforms in the modern age, where anyone can be a content creator. [6] Statistics from the popular social media platform TikTok which boasts over 1 billion monthly active users show that 83% of the platform's users have posted a video on the platform. [7]

The cardinal objective of a social media platform that aims to be successful and vibrant is an active and engaged user base. Achieving user engagement boils down to presenting the most attractive and relevant content to each user. However, popularity and success is a double edged sword since the abundance of users and content on these platforms floods users with huge amounts of information and hence poses a great challenge in terms of information overload. While search capabilities slightly alleviate the problem, often users are unable to express keywords that convey requirements about the type of content they would be interested in. Further, users tend to have diverse taste and the quality of content may be subjective depending on the user searching for it, therefore,

beyond simple searching capabilities, personalisation is also required to make the content attractive and relevant to each user. [8], [9]

A Social Recommender System is an intelligent system that filters the massive amounts of information on social media platforms and recommends useful items and information to users based on their personalised needs which are inferred through unique explicit and implicit interactions within the social network. In this way, Social Networks and their Recommender Systems tend to have a symbiotic relationship since the quality of recommendations catered to users allows the networks to grow in size, which in turn provides more interactions, allowing higher quality recommendations. [10]

GraphJet [11] is an example of a graph based Social Recommender system used for generating content recommendations in Twitter. GraphJet is able to provide personalized, real-time content recommendations for Twitter users, i.e. for a given user, it is able to recommend tweets that the user may be interested in based on the user's history and social interactions. To serve these recommendations, a personalized SALSA algorithm [12] is run on a bi-partite graph of interactions between users and tweets. The system assumes that the entire graph can be stored in the memory of a single server.

However, existing Social Recommender systems such as GraphJet are designed to work in traditional centralized social networks. In these centralized networks, users trust third party service providers (such as Meta/Twitter/Google) with their data and to provide them with recommendations. Recently, there has been an erosion in this trust due to violations of user privacy, whether it was intentional violations through the sale of data to third parties [13] or unintentional violations through the loss of data through hacking breaches in the platforms. [14]

This erosion of trust has led to a rise in popularity of "Web3", which leverages decentralized technologies such as distributed ledgers to offer decentralized alternatives of centralized platforms, such as social networks. [15] Web3 platforms allow direct interactions between users without any third-party intermediaries or centralized servers. This is achieved through utilizing communal infrastructure and resources provided by the participating individuals themselves.

In this paper, we address the challenges associated with generating recommendations in Web3 platforms by presenting Web3Recommend, a novel distributed, social recommendation system. Our approach integrates a graph-based content recommendation algorithm inspired by GraphJet with MeritRank [16], a Sybil tolerant feedback aggregation mechanism. By leveraging personalized SALSA algorithm with sybil-resistant random walks, we aim to strike a balance between trustworthiness and relevance in recommendations.

Web3Recommend contributes to the existing body of research by providing an end-to-end implementation that can be seamlessly integrated into any Web3 platform running on Android. We fully implemented the recommender to generate music recommendations for users of MusicDAO [17], an open-source Web3 music sharing platform which offers a decentralized alternative to Spotify/Apple Music.

First, we describe the problem that our system aims to solve in II, next we present the key features of of our solution in III. Then we provide some background on the concepts discussed and techniques used in our solution in IV, V and VI. Our system's model, assumptions and limitations are discussed in VII. In VIII we detail the implementation of the major components in Web3Recommend. The details on how we combined the two systems to generate our recommendations is presented in IX.

In XI we demonstrate the trust-relevance balance of recommendations through four sets of experiments. The first two experiments show that the recommendations generated are relevant and that the relevance of non sybil nodes is not too greatly diminished with increasing MeritRank decay parameters. The last two experiments involve adversarial Sybil attacks which allow us to demonstrate the sybil resistance of the recommender system.

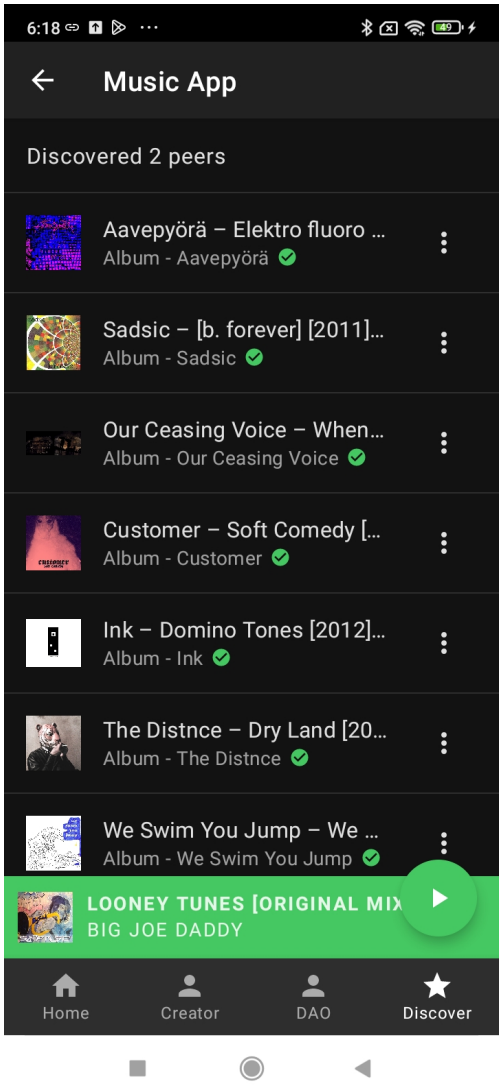


Fig. 1: Recommendations generated in MusicDAO

II. Problem Description

The generation of recommendations in Web3 platforms faces unique challenges compared to traditional centralized models. In conventional Social Recommender systems, a users' prior experience can be viewed as a vote in favour of certain items. Using a Sybil Attack [18], an attacker can create a potential unlimited number of fake identities (or sybils) to cast misleading votes.

Thus, by creating malicious users or identities, a bad actor could potentially mislead the system system into recommending their desired items. The problem becomes quite important when one considers the prevalence of abuse, fraud and spam on online social media platforms. Carrying out such an attack is not incredibly complicated, with existing solutions such as Tube Automator and Friend Bomber which make such attacks readily available to malicious users.

Recommender systems in centralized networks are relatively more secure from Sybil attacks due to the centralized nature of the system. Centralized systems often require user authentication and verification, making it more difficult for malicious actors to create multiple fake accounts to manipulate the ranking of items. Additionally, centralized systems have the advantage of being able to monitor user behavior and detect anomalies such as unusual activity patterns or highly repetitive actions, which could indicate the presence of a Sybil attack. This detection is possible because of the large number of skilled attendants dedicated to maintaining and improving system capabilities in centralized systems. These attendants can develop measures to prevent Sybil attacks, such as restricting the number of user actions that can be performed within a certain time frame, or introducing identity verification requirements.

Further, the decentralized nature of Web3 platforms, coupled with the pseudonymity and anonymity they offer creates an environment where Sybil attacks can occur more easily. Unlike in centralized systems, where user identities are typically verified, Web3 platforms prioritize user privacy and allow users to operate under multiple pseudonyms. This makes it challenging to establish the authenticity and credibility of users and their preferences.

Therefore, creating a trustable and reliable decentralized recommender system for Web3 platforms is a challenging task due to the lack of centralized infrastructure, making it easier for malicious users to create and control multiple identities, manipulate the ranking of items, and compromise the trustworthiness of recommendations. Therefore, the creation of decentralized recommender systems requires new approaches that can address the challenges of decentralized networks, including sybil attacks, limited resources, and lack of trust among users. [19]

III. Key Features of Web3Recommend

1) **Based on Monte Carlo-type methods relying on random walks**
   Web3Recommend uses personalized ego-centric random walks to perform computations of estimated Personalized Page Rank and SALSA values for nodes in the network. These values are then used to generate recommendations in the system. It has been shown that Monte Carlo methods can provide very good probabilistic estimations for Page Rank and SALSA. They are also much faster and parallelisable than the conventional power iteration method, making them a good choice for an online recommendation system. [20] Additionally, by enhancing the random walks in these methods with decay parameters from MeritRank, Web3Recommend is also able to limit the influence of sybils in these estimations. They also allow us to create a system that is simple and understandable, yet also sufficiently expressive to generate relevant, trustworthy recommendations. Random walks allow us to define a large design space, allowing room for customization to a wide domain of use cases in different applications (e.g. social search) and in different contexts. Further, random walks act as "social proof" for the recommendations, allowing users to better understand why certain items were recommended, leading to higher user engagement. There is also room for further increasing the quality of generated recommendations by feeding the output from our random walks as input to machine learning models, but in our case, the direct output is sufficient for user consumption.
2) **Each node stores the entire interaction graph between users and items which is synchronized using an edge gossiping mechanism**
   While readers might find the idea of storing the entire graph in a node strange, graph partitioning remains a complex problem in large, dynamically changing graphs despite a lot of work and progress in the field. [21] Achieving graph partitioning would require implementing a fully-distributed graph progressing engine and further add high communication cost to the system. Further, in a P2P Web3 platform, users are not expected to be always available/online, thus, similar to GraphJet, Web3Recommend assumes that all nodes store the entire graph in memory. Upto a billion edges can be reasonably stored in less than 8GB of memory with our compact graph serialization techniques which is a reasonable assumption to make given the hardware present in modern devices.
3) **Bootstrapping mechanisms**
   The "new user" problem in Social Recommender system [22] necessitates the need of bootstrapping mechanisms to introduce new users to the network. Web3Recommend includes two bootstrapping mechanisms: 1) A similarity based mechanism for finding similar users, allowing new users to find users they can trust 2) A personalized page rank for creating a circle of trust which can be used for recommending relevant items to users who haven't consumed many items and thus don't have any existing edges in the interaction graph

IV. Background on Trust

The system presented in this paper relies on the (incremental) computation of personalized PageRank and SALSA augmented with principles from MeritRank. We also build on top of the GraphJet recommender system by Twitter. In this section, we provide a quick review of these methods.

A. PageRank

One of the most widely known ranking systems in the world is Google's PageRank [23] which is still used (along with other algorithms) in order to rank websites for user queries on Google.

PageRank determines a rough estimate of the relative importance of a website by computing a ranking for every web page. The underlying assumption of PageRank is that a website that is more important is more likely to receive links from other websites than a website that is less important i.e. that the existence of a hyperlink $u \to v$ implies that the page u votes for the quality of page v and hence, the most important page is one that receives most votes. PageRank gave birth to the idea of topic sensitive or personalized ranking and other hyperlink-based centrality measures. [24]

More formally, let V represent the set of all web pages in a network. A *web graph* is the directed graph which consists of the vertex set V and the hyperlinks between pages represent the edges in the network. Further, let r be the *preference vector* which induces a probability distribution over V and $c \in (0, 1)$ be the *reset probability*.

Then, the PageRank vector p is the solution of the following equation: [25]

$$p = (1 - c) \times pA + c \times r \tag{1}$$

If r is uniform over V then p is referred to as the *global PageRank vector*. In this paper, the special case where for some web page x, the x^{th} coordinate of r is 1 and the rest of the coordinates are 0, the solution of r represents the *Personalized PageRank* of web page x and is denoted as PPR(x).

Note that it is also equivalent to interpret PageRank with the Random Surfer model [26] where PageRank is simply the stationary distribution of a random walk where at each step, assuming we are at a certain web page u, with probability c we jump to a random web page v, and with probability $1 - c$ we follow a randomly chosen outgoing edge (or hyperlink) from u to a new web page w. In this model, Personalized Page Rank is the same as PageRank except for the difference that all random walks start and jump randomly to the seed node x for which we are personalizing the PagenRanks. [27]

B. SALSA

Stochastic Approach for Link-Structure Analysis or SALSA is a web page ranking algorithm similar to PageRank and HITS [28] which attempts to extract information from the link structure of a networked environment to associate two scores with every node v, the **hub score** h_v and **authority score** a_x. As the name suggests, a_x reflects how much of an *authority* a node is on a certain topic. While the notion of authority is pretty broad, the intuitive idea is that in the context of a specific query, links from one node to another express a considerable amount of latent human judgement and the that judgement is precisely what is needed to formulate the notion of authority. Hence, if many nodes point to another node it will possess a high authority score. h_v on the other hand reflects how well nodes point to authorities, so if a node largely links to other nodes that are considered very authoritative, it will have a high h_v.

More formally, if E is the set of all edges in the graph and $indeg(x)$ and $outdeg(v)$ are the in-degrees and out-degrees of the node respectively then:

$$h_v = \sum_{\{x|(v,x)\in E\}} \frac{a_x}{indeg(x)} \tag{2}$$

$$a_x = \sum_{\{v|(v,x)\in E\}} \frac{h_v}{outdeg(v)} \tag{3}$$

It is worth noting that unlike PageRank where we only have forward random walks, SALSA consists of forward-backward random walk where the walk alternates between forward and backward steps.

Similar to the notion of Personalized PageRanks, we also have Personalized SALSA which tailors the hub and authority scores to a single root node. As in Personalized PageRank, in the personalized version of SALSA, we can have random jumps to the seed node. Assuming that the seed node is u, $h_{v,u}$, the personalized SALSA hub score and $a_{x,u}$, the personalized SALSA authority score can be represented as:

$$h_{v,u} = c \times \delta_{u,v} + (1 - c) \times \sum_{\{x|(v,x)\in E\}} \frac{a_x}{indeg(x)} \tag{4}$$

$$a_{x,u} = \sum_{\{v|(v,x)\in E\}} \frac{h_v}{outdeg(v)} \tag{5}$$

Notice that this in this setting, the hub score and authority score can be interpreted as the similarity and relevance score respectively. This ability of personalized SALSA to create tailored recommendations for specific root nodes is used later on inside our recommendation system.

C. GraphJet

GraphJet is a graph-based system for generating real-time tweet recommendations at Twitter. The recommendation algorithm is based on an adaptation of personalized SALSA which involves random walks in a bi-partite graph of users and tweets. Formally, GraphJet manages a dynamic, sparse, undirected bipartite graph $G = (U, T, E)$ where U represents users in Twitter, T represents tweets and E represents interactions between the users and tweets over a temporal window. Hence, the bipartite graph G consists of two set of nodes U and T, users from U are always on the left side of the random walk and represent *hubs*, while tweets from T are on

the right side and represent *authorities*. Figure 2 demonstrates a sample bi-partite graph used by GraphJet, where the left side consists of users from the "circle of trust" of the user whose recommendations are being generated and the right side consists of tweets the users in the circle interacted with. The circle of trust is constructed using a personalized PageRank algorithm.

GraphJet maintains and updates the bipartite graph by keeping track of user-tweet interactions over the most recent n hours. Periodically, edges older than n hours are discarded to ensure that memory consumption doesn't increase boundlessly, experiments show that this pruning does not have a noticeable impact on the system's recommendation quality. The system supports high performance ingestion of real-time and interactions and generations of recommendations.

Below is a simplified description of the SALSA algorithm run inside GraphJet:

1) The random walk begins from the vertex u in the left hand side of the bi-partite graph corresponding to the query user
2) An incident node from u to the right hand side of the graph is uniformly selected to a tweet t on the right hand side of the bi-partite graph
3) From t, an incident edge is selected uniformly back to the left hand side to another node v
4) This is repeated an odd number of steps

Figure 3 demonstrates a sample random walk in one iteration of the above SALSA algorithm. To introduce personalization, a reset probability as described in IV-B above is used, which restarts the random walk from vertex u to ensure that the random walk doesn't "stray" too far from the query vertex. Further, it may also be possible that the query user doesn't have any existing interactions in the bi-partite graph, either because the last interaction was more than n hours ago or because they are a new user. In this case, the random walk could start from a *seed set* instead of from the query user's node. This seed set is configurable but the usual chose is to use the circle of trust of the user constructed using Personalized PageRank.

After constructing the bi-partite graph, multiple instances of the SALSA algorithm are run which assigns hub scores to the left side and authority scores to the right side. The vertices on the right hand side are then ranked and presented as tweet recommendations to the user. The vertices on the left hand side are also ranked and based on the homophily principle can be presented as "similar user" recommendations.

The GraphJet paper suggests that the algorithm is effective because it is able to capture the recursive nature of the user recommendation problem. A user u is bound to also like tweets that are liked by users that are similar to u. These users are in turn similar to u if they follow the same (or similar) users. Personalized SALSA operationalizes this idea, providing similar users to u on the left-hand side and tweets that they like on the right-hand side. The random walk in SALSA also ensures equitable distribution of scores out of the vertices in both directions.

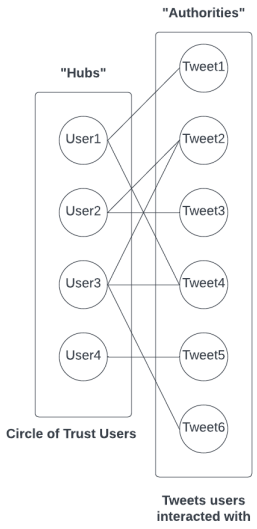


Fig. 2: Example of bi-partite graph used in GraphJet

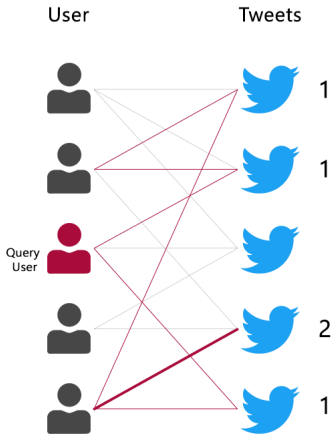


Fig. 3: Example of a random walk in GraphJet's Personalized SALSA

V. Background on Decentralisation

A. Decentralisation

The term "decentralised network" was initially introduced by Paul Baran, one of the pioneers of packet switching. Networks can generally be classified into two types: "star" or centralized networks and "grid" or distributed networks. In a star/centralized network, all nodes are connected to a central node, requiring participants to go through this central component to interact with one another. In contrast, distributed networks have no central node, enabling direct communication between nodes without reliance on a centralised point. Baran termed networks that utilized a combination of these components as "decentralised" since they lacked a single, central

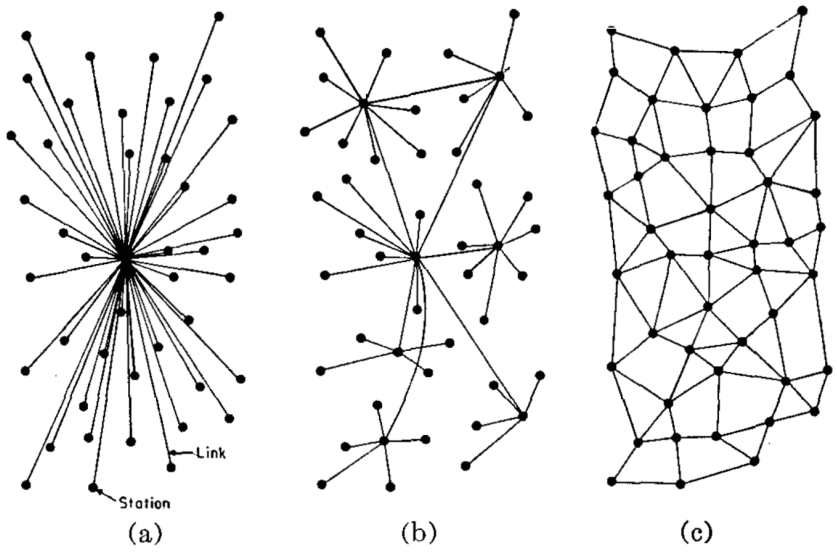


Fig. 4: a) Centralised network b) Decentralised network c) Distributed network
Cardinal architectural insight from Baran's 1964 paper [29]

point of failure. [29] Figure 4 visually illustrates these network types.

In contemporary literature, the term "decentralised network" refers to networks where participants and contributors control the technology, content, and infrastructure, rather than relying on large central platforms. This control manifests in various ways, such as participants managing specific parts of the infrastructure (e.g., servers and routers), collaborators owning their own private data silos that are queried during network discovery, and participants having autonomy in determining the operational details of the network, including content publicity and deletion decisions. [30] Twitter serves as an example of a centralised network, where the platform owns all user-generated content. Conversely, Tribler, a peer-to-peer file sharing system that builds upon the BitTorrent protocol, exemplifies a decentralised network. Tribler enables users to share content using keyword search and incorporates a reputation-management system to foster collaboration. [31]

B. Web3

The term "Web2.0" was originally introduced by Tim O'Reilly in 2007 to describe a new iteration of the Internet that empowered users to publish, consume, and interact with content and each other. It aimed to expand upon the earlier version, "Web1.0," which primarily featured static pages for information display. While "Web1.0" was often referred to as the "read web," "Web2.0" aimed to be the "read-write web," a term coined by Richard McManus in 2003.

Critics, including Tim Berners-Lee, the inventor of the World Wide Web, argue that "Web2.0" failed to fulfill the vision of a secure, decentralised exchange of public and private data. Instead, users' data became increasingly stored in corporate data silos, raising concerns about data ownership and security. Berners-Lee and others advocate for users to have ownership of their own data to ensure data security.

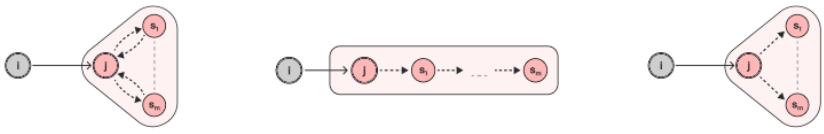
In 2014, Gavin Wood, co-founder of Polkadot and Ethereum, introduced the term "Web3.0" to describe an Internet that is decentralised, open, and transparent. The Web3 movement seeks to transform the platform-oriented "Web2.0" into a decentralised web ecosystem with the following goals: 1) avoiding content discovery and propagation monopolies by large centralised entities, 2) combating the spread of misinformation and fake news, 3) enabling users to create, exchange, and react to information securely, privately, and freely, and 4) supporting immersive web development.

Liu et al. define Web3 as a movement that goes beyond specific applications or underlying infrastructures, aiming to establish "an era of computing where the critical computing of applications is verifiable." In other words, an application aligning with the Web3 idea allows all stakeholders to verify its execution based on predetermined terms without the need for intermediaries.

C. MeritRank

MeritRank [32] aims to bound the benefits of Sybil attacks instead of preventing them altogether. The system is based on the assumption that peers observe and evaluate each others' contribution. Each peer's evaluation is stored in a personal ledger and modelled in a feedback graph where the feedback to each user is modelled as a special token value which accumulates over time. It is also assumed that each peer is able to discover the feedback graph, for example, through a gossip protocol.

In order to limit the influence of Sybils, three main types of Sybil attack strategies are identified in MeritRank, these are illustrated in Fig. 5.



(a) Cycle attack (b) Serial attack (c) Parallel attack

Fig. 5: Sybil attack strategies [32]

MeritRank manages to achieve Sybil tolerance by imposing the following constraints on how reputation can be gained inside the feedback graph when using any of the above strategies:

1) **Transitivity α decay**
   This constraint limits the ability of an entity to create a serial Sybil attack by terminating random walks in the feedback graph with a probability α
2) **Connectivity β decay**
   Sybil attack edges in a feedback graph are often bridges i.e. their cut creates two separates components. This constraints introduces a punishment for a node for being in a separate component

A trust graph modelled using these MeritRank's constraints will satisfy:

$$\lim_{|S|\to\infty} \frac{w^+(\sigma_s)}{w^-(\sigma_s)} \leq c \tag{6}$$

where, $w^+(\sigma_s)$ is the profit gained by the Sybil Attack σ_s, $w^-(\sigma_s)$ is the cost of the Sybil attack, S is the set of Sybils and c is some constant value such that $c > 0$. Thus MeritRank is able to provide a reputation system with feedback which is Sybil tolerant.

VI. Related Work

In this section, we cover broad categories of existing work in research which attempts to solve similar problems and show why they do not achieve the goals that we attempt to fulfill. Thus, we demonstrate the relevancy of our work by pointing out how it achieves the interesting niche of generating trustworthy and relevant recommendations in Web3 platforms.

A. Bounding Identity Creation

A popular method to defend against sybil attacks is to leverage defenses that bound the ability of malicious attackers to create sybil identities and hence indirectly limit the influence of the sybil attack by limiting the votes that a sybil attack can cast. The most rudimentary method of achieving this is by ensuring that all unique identities on platforms correspond to real human beings. This can be achieved using a trusted central authority to verify information about users which is unique to human beings, such as passports, phone numbers, credit card etc. Other approaches involve using graphical challenges such as CAPCHA to ensure that the user is a human. [33] Such approaches are flawed in the context of Web3 for multiple reasons. First, users of Web3 platform employing such methods may be hindered from using them because of privacy concerns, the anonymity guarantees of these platforms is often the primary reason many of the users choose to use these platforms. Second, using a centralized third party for verification is antithetical to the idea of decentralization that Web3 platforms stand for and further, maintaining a centralized server adds too much complexity to the design of the system.

Decentralized approaches to the same problem have been suggested in research such as limited identities from certain IP addresses/prefixes [34], creating resource based challenges [35] and remote issuing of certificates to verify identity based on network/location coordinates [36]. While these approaches do make it harder to create sybil identities, they do not entirely stop sybil attacks and a powerful adversary which enough incentive could easily surpass these mechanisms to generate significant influence on the recommendations in the network. Further, bounding identity creation can also have the uninteded effect of make it frustrating for non-sybil users to use the platform. Therefore, our work does not rely on bounding user creation to indirectly limit sybil influence and instead directly limits sybil influence by effectively detecting sybils and restricting the influence of their votes.

B. Reputation Systems

Reputation Systems [37] allow the collection of feedback from different users in the network to determine which peers can be trusted based on their past behaviour. A popular example of a reputation system is the "Feedback Forum" on Ebay [38] : after a transaction is completed, a buyer or seller can rate each other (1, 0 or -1) and leave comments. A participant in eBay accumulates such points over time which are displayed next to their screen name. A buyer can view a seller's points and comments left by other users to create a "shadow of the future" into the transaction they can expect to have if they buy an item from the seller. Many other online forums and marketplaces such as Amazon and Stack Overflow rely on similar reputation systems. However, while reputation systems are a strong mechanism to determine whether certain users are trustworthy, on their own they do not provide a way to generate recommendations based on trustworthy users.

C. Social Network based Sybil Defense

Other works have utilized the power of feedback from social networks to limit sybil influence. Popular examples are SybilGuard [39], SybilLimit [40], Ostra [41] and SumUp [42].

Similar to our work, these methods are able to establish an approximate notion of trust among users using the properties of graphs and often assume global knowledge of the dynamically changing social network. Using these properties, the influence of sybil identities in the votes is limited. However, similar to vanilla Reputation Systems, while these approaches are great for generating trust in the network, on their own they are not able to ensure that the non-sybil items they identity are relevant to the user who the recommendations are being generated for. We are able to achieve this by building on personalized SALSA and ensuring that not only is the recommended item trustworthy (i.e. non-sybil), but also relevant to the user the recommendations are being generated for.

D. Machine Learning based approaches

Existing machine learning based approaches to recommender systems attempt to apply techniques from the multiarmed bandit problem [43] or the contextual bandit problem [44] where contextual information is used to group users that belong to the same cluster via classification or clustering techniques. The problem with this approach is that they assume the presence of considerable existing feedback from users on what items they like and on the "goodness" of various objects, since these models are only as good as the data that they are trained on. Web3 is an emerging technology and in a lot of platforms (especially new platforms) this data is missing or not sizable enough to allow the training of decent models, leading to poor recommendations or the inability to have such models altogether. [19]

E. Other approaches

Another notable paper worth mentioning is Dsybil [19], the system presented in the paper utilizes similar mechanisms for diminishing sybil influence and generating recommendations as our paper. They are able to achieve this by: i) exploiting the heavy-tail distribution of the typical voting behavior of the honest identities, and ii) carefully identifying whether the system is already getting "enough help" from the voters and hence, if sybil votes are only latching on to existing votes. While they demonstrate an impressive sybil tolerance, a notable drawback of their paper is its reliance on only explicit feedback through voting on items that need to be recommended. However, explicit feedback is not always available and in fact, most of the feedback on social networks is implicit rather than explicit. [45] Our system incorporates implicit feedback from users in order to generate recommendations. Further, like a lot of other mentioned papers, the system focuses purely on trust and not on generating relevant, personalized recommendations.

VII. System Model, Assumptions and Limitations

A. Items, Users and Votes

Web3Recommends recommends *items* (e.g. songs/albums in Spotify, movies in Netflix, posts in Reddit etc) to the platform's *users* based on the past experience of the users with the items. A user's preference for an item serves as a *vote* for the item, therefore, if enough users like a certain item, it is likely to have more votes by virtue of being visited more in random walks inside the personalized SALSA algorithm and therefore, more likely to be recommended to other users.

B. Target Application/Scenario

Recommendation Systems are a very broad concept and different systems differ in their goals and details. [46] Therefore, a solution that works in one scenario may not work in another scenario because the platform requires a different purpose from its Recommendation System. For example, a Recommendation System in an online retailer such as eBay may be required to generate all products that a user may be interested in so that the user has multiple choices while the system in modern media applications such as TikTok would only be required to generate a single recommendation to autoplay as the next item for the user.

Web3Recommend aims to cover a broad range of use cases by providing a system that is able to provide a ranking of all items available to a particular user. For this, it assumes scenarios where:

1) The objects to be recommended can be uniquely identified and are always available to the user (the exact method of availability could vary from peer-to-peer to being provided by a central server)
2) The lifespans of the users in the network is not incredibly short lived, allowing them to establish trust relationships with other users
3) Users are able to initially discover trusted users using social discovery mechanisms or an initial set of trusted users are provided to the user by the platform

The final assumption is a significant limitation of our work since it's impossible to build trust out of nothing and it is possible for a new user to end up trusting sybil users via bootstrap threats.

C. Network Assumptions

Web3Recommend assumes that the application utilizing it leverages a peer to peer architecture [47] with each user operating their own node and possessing the ability to communicate directly with any other node and item in the network. We assume that all users in the network follow the protocol honestly and cannot tamper with it in anyway. We also assume that communication occurs over privacy preserving, cryptographically secure protocols.

While the system presented in this paper relies on Gossiping VIII-C over a content overlay network to synchronize its data, it is important to acknowledge that it currently lacks sufficient security measures to protect against spoofing of gossip. However, it is worth exploring the potential of enhancing the system's security through the implementation of a certificate-based approach. By utilizing certificates for each node within the content overlay network, it becomes possible to establish a secure communication framework. These certificates can be used to verify the authenticity and integrity of gossip messages exchanged between nodes, ensuring that only

trusted and authorized nodes participate in the synchronization process. Moreover, by incorporating individual certificates for each user, the system can prevent users from pretending to be another user and signing actions on their behalf. This significantly raises the bar for spoofing attempts, enhancing the overall security of the gossiping process.

D. Affinity and Trust

Web3Recommend uses two distinct concepts, *affinity* and *trust* to model user to user and user to item relationships respectively.

A user u's affinity for item i is expressed by $Af(u, i)$ and is calculated by:

$$Af(u,i) = \frac{PC(u,i)}{\sum_{x \in I_u} PC(u,x)} \tag{7}$$

Where $PC(u, i)$ is the *play count* of user u for item i i.e. the number of times the user has consumed item i and I_u is the set of all items that have been consumed by user u. Therefore, $Af(u, i)$ is a value between 0 and 1 and serves as a measure of the user u's preference for item i compared to all the items that they have consumed.

This definition of affinity may not suit all use cases and a more fine grained metric such as share of total play time could be more appropriate.

Like in reputation systems, a user's trust in another user is increased when the user performs useful work for the other, which in this case of our recommendation systems means providing a recommendation that the other user likes i.e. has a high affinity for. Therefore, affinity and trust are inherently linked since as a user's affinity for an item increases, the user's trust in other users who recommended the item to them also increases.

Since in our system recommendations are calculated using random walks inside a personalized SALSA algorithm, recommending an item to another user means that the recommender "voted" for the item in the random walk by having a high affinity for it. Note that modelling trust this way ensures that sybil users cannot simply increase other user's affinity in their sybil items by having a high affinity for popular items since: 1) if they have a high affinity for popular items, their sybil items are less likely to be recommended 2) Many users are likely to have a high affinity for popular items thus sybils are less likely to benefit from following this strategy 3) In order to gain trust, the sybils still need to effectively perform "useful work" by recommending items that other users like. They do not simply reap rewards from items that have already been recommended to other users.

VIII. Web3Recommend Architecture and design

Web3Recommend is a Recommender System designed to provide recommendations in any application running on a distributed network. The central data structure in the network is the **TrustNetwork** which stores information about user to user and user to item relationships across the entire network. Recommendations in the system are generated by performing random walks inside this network. Each node maintains a personal copy of a TrustNetwork and updates to the network are synchronized through a timestamp biased edge gossiping mechanism which ensures that recommendations are based on recent, global information inside the network. The system design also includes a simple bootstrapping mechanism which allows new users to find similar users in the network, however, it is worth noting that this bootstrap mechanism can be exploited by malicious users and in a real application, we assume that users are able to bootstrap through social discovery of trusted peers or through the provision of trustworthy nodes by the application itself. The following is an in-depth description of the various components of the system:

A. TrustNetwork

As mentioned before, the TrustNetwork is the central data structure of Web3Recommend. TrustNetwork consists of two different types of nodes: users, U and items, I. Further, there are two different types of edges in the network: directed user to user weighted edges which represent the trust a user places in another user and undirected user to item weighted edges which represnt the affinity of a user for an item. In practice, this is implemented using a combination of two graph strucutres:

1) **User to User Graph**
   The user to user graph is a *weighted directed acyclic graph* in which the vertex set of the graph consists of all users in the network and the edge set consists of trust relationships between the users. In order to ensure efficient memory usage and to guarantee that an edge between two users only exists if they trust each other, only the top 5 edges in terms of trust/weight are retained for each user.
2) **User to Item Graph**
   The user to item graph is a *weighted undirected acyclic bi-partite graph* in which the vertex set of the graph consists of all users and items in the network and the edge set consists of affinity relationships between the users and items

All algorithms in Web3Recommend operate on top of the TrustNetwork. In our Kotlin implementation, we use JGraphT [48] for the efficient implementation of both the graph data structures.

B. Recommendation Algorithm

The main component of Web3Recommend is its elegant recommendation algorithm. The algorithm is inspired by GraphJet's algorithm presented in IV-C.

We present three modifications to the personalized SALSA used in the original algorithm.

1) **Weighted Random Walks**
   Instead of using uniform probabilities to decide which node to walk to in our random walks, we perform walks using the affinity metric defined earlier. Hence, when walking from a node to an item the walk is biased by the affinity for the item, so if a user u prefers an item i_j two times more than item i_k, the random walk is also

twice more likely to visit i_j from u than i_k. We believe this is a reasonable assumption which aligns with our goal of recommending items that similar users like. Similarly, when walking back from an item to a user, the walks are biased by the user's affinity for the item. Hence, we are more likely to travel to a user who has a higher affinity for the item. Again, finding similar users boils down to finding users who like the same items so it's reasonable to bias our walks this way.

2) **Add MeritRank decays to limit the influence of sybil attacks**
   In order to add sybil tolerance to the system, alpha and beta decays from MeritRank are added to the system. In IX we explain how the decays are calculated and used to provide sybil resistant recommendations. In our experiments, we vary these decay values to measure their influence on trust (sybil tolerance) and relevancy of recommendations.
3) **Generate "trusted" random walks**
   While GraphJet's personalized SALSA algorithm allows the generation of personalized, relevant recommendations, the recommendations generated are not guaranteed to be trustworthy. This is because when performing a random walk from an item to a user, it's possible to walk to a non trusted (sybil) user who also claims to like the item. In Web3Recommend, we modify the walk back from an item to a user to be limited to users who the original voter of the item trusts. This ensures that if we reach a sybil user in a random walk, it's through a malicious user or another sybil user and hence, their influence on voting can be detected and limited through MeritRank's decays.

A simplified version of a random walk in our personalized SALSA algorithm is illustrated in 6.

C. Timestamp Biased Edge Gossiping

Since Web3Recommend relies on each user storing a local copy of a $TrustNetwork$ each, a mechanism for synchronizing the $TrustNetwork$s across different users is required. Gossiping [49] has proved to be a successful mechanism for supporting dynamic and complex information exchange among distributed peers. Gossiping based mechanisms are great for building and maintaining the network topology itself as well as supporting a pervasive diffusion of the information injected into the network. [50] Gossiping takes inspiration from the human social behavior of spreading information through peers who are in direct contact.

Our gossiping mechanism relies on randomly gossiping user to user edges and user to item edges to other users in the network. In addition to gossiping, we leverage Semnatic Overlay Networks (SON) [51] which guarantee that the users we gossip to share similar interests.

GraphJet only generates recommendations based on recent interactions between users and tweets. In order to achieve this, only the n latest interactions in the network between users and tweets are stored. Doing this has two advantages:

1) It allows the creation of temporally contextual (real-time) recommendations which have have a profound impact in retail, media, entertainment and other contexts [52]
2) It provides a mechanism for limiting the memory usage on a device by using n as a tunable hyper-parameter, thus allowing the recommendation algorithm to also be feasible in resource limited devices such as older smartphones

Web3Recommend achieves this by adding timestamp to all edges and removing older user to item edges after the total number of user to item edges in the network exceeds n.

Note that while user to item edges are deleted after a time window, user to user edges are persisted and only deleted either: 1) When they receive gossip with a more recently created version of the edge or 2) The user to user edges of the user exceeds a threshold in which case the edge with the smallest weight is deleted. We believe that this is reasonable since in most media platforms the amount of items greatly exceeds the number of users and hence the memory cost of this design choice should be low. Further, it allows us to persist long-term trust relationships between users which are used in VIII-D.

We also want to ensure that newly created edges have a higher chance of being gossiped, in order to do this we construct a mapping of edges to their delta from the oldest edge in the network. Then, the array is softmax which produces weight values which provide the bias by which an edge is gossip, hence, a newer edge is much more likely to be gossiped than an older one.

D. Bootstrap

Recommender Systems suffer from the "cold start" problem where the systems meets a new user for the first time. Since the system has no history of the user's interactions, it can't establish the user's personal preference. In a real-time recommendation system such as ours this problem is compounded since an inactive user querying for recommendations may not exist in the interaction graph.

1) Circle of Trust: GraphJet solves this problem by starting the random walks from a **seed set** instead of a single node. The seed set could be provided by the network or be constructed from the user's trusted users called the "circle of trust".

In a social network like Twitter this circle of trust can be calculated using the user's social connections/follows. In our system, we calculate the circle of trust using an Incremental Personalized Page Rank [27] algorithm that ranks nodes in the network in order of their trustability while ensuring that subsequent random walks are incrementally computed.

Since unlike user to item edges, user to user edges are persisted over time, inactive users can still be served personalized recommendations in this manner.

We implement the selection inside the seed set in a dynamic manner that ensures that as the user has more user to item interactions, random walks are more likely to start from the user rather than the seed set. Hence, it serves as a bootstrap measure to acquaint the user to the network.

2) New User: The previous bootstrap mechanism relied on the user having prior interactions or social trust in the network. For a fresh user neither of these exist and hence a different bootstrap mechanism is required to acquaint them to the network.

For this we provide a User Collaborative Filtering algorithm based on the work from [53] for finding similar users in the network. Initially, users are provided with "divisive" content in the network which can be used to find more information about their taste profile. This helps to establish user to song edges which can be used to measure similarity between other users.

The similarity $sim(a, b)$ between two users $(a, b) \in U$ where U is set of all users is calculated as:

$$sim(a, b) = Nsim(a, b) \times \tau + (1 - \tau) \times Dsim(a, b) \quad (8)$$

Where $Nsim(a, b)$ is calculated as:

$$Nsim(a, b) = cf(a, b) \times sim(a, b) \quad (9)$$

Here, $sim(a, b)$ is the Pearson Correlation Coefficient [54] of the two user's item ratings calculated as:

$$sim(a, b) = \frac{\sum_{i \in I_{ab}} r_{a,i} r_{b,i}}{\sqrt{\sum_{i \in I_{ab}} r_{a,i}^2} \sqrt{\sum_{i \in I_{ab}} r_{b,i}^2}} \quad (10)$$

Where $I_{a,b}$ is the set of common rated items between both users a and b. $r_{a,i} = R_{a,i} - avg_a$, $R_{a,i}$ is the rating given by user a to item i and avg_a the average of all ratings by a.

$cf(a, b)$ is the *common preference degree* between user a and b:

$$cf(a, b) = \frac{|I_a \cap I_b|}{\max_{x \in U} |I_a \cap I_x|} \quad (11)$$

Where $|I_a \cap I_b|$ is the count of common rated items between users a and b.

Hence, $Nsim$ 9 uses the traditional measure of similarity between users modified by the degree of common preference between the two users.

$Dsim$ is a measure of similarity calulated using the rating difference of users on their common items. It helps calculate a more fine tuned similarity metric than coarsely comparing items both users have interacted with. The exact formula has been omitted for space limtations, the code implementation can be found here.

Hence, $sim(a, b)$ is used to compare a similarity metric to other users in every user and hence serves as a bootstrap measure for new users.

Note that this strategy requires that the users being introduced to the new user are non sybil. If the strategy is run naively against all users it is prone to bootstrap threats as mentioned in VII-B.

E. Compact Serialization

Since Android devices often have limited resources, it's important to be able to store networks compactly. Once a user disconnects from the network, the *TrustNetwork* is serialized and stored on the device.

Our design is inspired by the DIMACSFormat in JGraphT which is based on the format used in the 2nd DIMACS challenge [55].

The details of the serialization have been omitted due to size limitations but the implementation is self documenting, rigorously tested and can be found here.

IX. Using MeritRank Decays to generate recommendations

As covered in V-C, MeritRank provides two main mechanisms for sybil-resistance, Alpha and Beta decays. Each of the decays is tunable and used to limit the gains of a Sybil Attack.

A. Alpha Decay

The alpha decay works exactly like the *reset probability* described in IV. The random walk in the user interaction graph stops with probability α. Setting an alpha decay reduces the effectiveness of sybil attacks that rely on long walks that get stuck inside a segment of sybil users as described in [32].

B. Beta Decay

Beta Decay punishes sybils for being isolated from the rest of the network. In our system, this is achieved by measuring the diversity of users voting for a song. Assuming the system's beta decay is set to β, the item beta decay $b[i]$ is calculated for each item $i \in I$ with the following formula:

$$b[i] = \begin{cases} 1 - \beta & \text{if } \exists u \in U : div(u, i) > \tau \\ 1 & \text{else} \end{cases} \quad (12)$$

Where U is the set of all users, τ is the beta decay threshold and $div(u, i)$ is defined as:

$$div(u, i) = \frac{\sum_{r \in R(i)} \begin{cases} 1 & \text{if } \exists u \in U : (u \in r) \cap (r[u] < r[i])) \\ 0 & \text{else} \end{cases}}{|R(i)|} \quad (13)$$

Where $R(i)$ is the set of all random walks that contain item i. Note that SALSA random walks can contain both users and items. $r[x \in U \cup I]$ is the index of a user or item in a random walk, hence, the step in the random walk when it was walked to. So, for each vote that item i received in a random walk, it measures what percentage of times user u led up to the vote.

Therefore, in 12 we measure the diversity in recommenders leading up to a vote and compare the "sybilness" of item i to the threshold τ and if it's deemed sybil, it is assigned a beta decay of $1 - \beta$.

The rankings of the items in the system are performed using a modified aggregation of random walks. A personalized

ranking score $s[i]$ is calculated for each item $i \in I$ using the following formula:

$$s[i] = \frac{|R(i)|}{\sum_{x \in I} |R(x)|} b[i] \tag{14}$$

The highest ranked items are then presented to the user as recommendations.

In our implementation, we provide two methods of calculating beta decays, one relatively faster option which gains its speed by iterating through each random walk linearly and precomputing the number of times a user is involved in a specific vote for each item. This is relatively fast and can be performed in linear runtime relative to the size of random walks. But the space cost of the algorithm makes it prohibitively expensive on devices with limited memory, hence, we also provide an on-the-fly implementation which is more time consuming but also with a tiny memory footprint.

X. Experiment Setup

A. Dataset and Test Network Generation

In order to demonstrate sybil resistance, it was important to create a network of non-sybil users on which we could mount sybil attacks. We used the dataset from the taste profile subset of the Million Song Dataset [56], [57] in order to generate this non-sybil network. The original dataset was sourced from The Echo Nest, which is an online resource that provides music applications on the web, smart phone etc. Therefore, using this data from real music platforms it was possible for us to generate a network that emulates the behavior of non-sybil users in Web3Recommend. The taste profile subset provides us with:

1) Users
2) Songs
3) User-Song-Play Count Triplet

For the test network generation, user-song edges were created by using the user-song-play count triplet to estimate user-song affinity as described in VII-D. User-User edges were created using the same user based collaborative filtering method we used to bootstrap trust relationships for new users. This is reasonable since all the users in this network were assumed to be non-sybil and hence, their trust between them is simply a function of their similarity.

Figure 7 shows some statistics of the test network.

B. Metrics used for experiments

1) Top x Ranking: The goal of Web3Recommend is to rank items available to a user in order of increasing relevance and trust to them. However, in most consumer driven recommendation systems, only the top few recommendations are important since those are the only ones the user would usually look at to choose the next item to consume. Therefore, in most of our experiments we measure the influence on the top 100/1000/10000 ranked items in the system.

Specifically, for the experiments which measure sybil resistance we were interested in measuring how many sybil songs ended up in these rankings. For example, a sybil item being the highest ranked recommendation to a user is much worse than two sybil items which are ranked much lower. Therefore, we define a metric *Sybil influence of top x ranking* or $SITR(x)$, which measures the influence of sybils in the top x category of ranking using the follow formula:

$$SITR(x) = \sum_{i=0}^{i=x} \begin{cases} x - i & \text{if } DRankedI(i) \in \mathbb{S} \\ 0 & \text{else} \end{cases} \tag{15}$$

Where, $DRankedI$ is the array of all items sorted in descending order of their ranking and $\mathbb{S}$ is the set of all sybil items. Hence, for example, if there is only one sybil item in the network but it is the highest ranked item, the $SITR(100)$ will be 100.

2) Ranked Biased Overlap: In order to measure the effect of decay parameters on the ranking of the items, we use Ranked Biased Overlap [58] between two sets of ranking to compare how similar they are. Ranked Biased Overlap is specifically constructed to be a similarity measure between incomplete ranking lists and hence fits our use case very well. RBO outputs a value between 0.0 and 1.0, where 1.0 represents two identical sets while 0.0 represents completely different sets.

XI. Experiments

A. Leave one out cross validation

In this set of experiments, we follow the below process:

1) 100 users from the User to Song graph are sampled randomly
2) For each user, we remove an existing user to song edge
3) We run the personalized SALSA algorithm with alpha decay value set to α
4) We measure: 1) The ranking score of the song whose edge was removed 2) In how many percent of trials the missing song shows up in the Top 100/1000/10000 rankings

While the goal of our paper is focused on limiting sybil influence, this experiment helps demonstrate that the foundational recommendation algorithm can provide reasonable personalized recommendations. The result of our algorithm is also compared to the result of a simple vanilla personalized SALSA implementation to show that our algorithm performs just as well if not better in terms of recommendation for our dataset.

B. Effect on ranking of increasing decays

In this set of experiments, we follow the below process:

1) 100 users from the User to Song graph are sampled randomly
2) For each user, we run the personalized SALSA algorithm for a specific alpha decay value $\alpha \in 0.1, 1.0$/beta decay value $\beta \in 0.1, 1.0$
3) For each ranking with increasing α or β values, we calculate the Ranked Biased Offset compared to the original ranking

This experiment helps to measure the influence on ranking with increasing decay values, showing us how increasing decays in the network and hence, increasing trust by reducing the influence of sybils impacts the relevance of recommendations. This could also be interpreted as the "false positive" rate of sybil detection.

C. Single sybil attack

As mentioned earlier, MeritRank provides sybil resistance by limiting sybil influence through:

$$\lim_{|S|\to\infty} \frac{w^+(\sigma_s)}{w^-(\sigma_s)} \leq c \tag{16}$$

In this experiment we prove this property in our network by constructing multiple sybil attacks where an adversary with an existing trust edge to non-sybil users starts mounting a sybil attack by creating multiple sybil identities. Thus, if we can prove that after a certain threshold of identities, the benefit to the adversary of creating more sybils is negligible, we can show that our system offers sybil resistance.

Therefore, in this experiment be follow the below process:

1) In our test network, we randomly sample a user
2) A trusted neighbor of the user is converted to a **traitor** who mounts either a: 1) linear sybil attack or 2) parallel sybil attack as defined in V-C
3) The traitor mounts attacks with increasing number of sybils and for each attack we measure the gain in total ranking score in our personalized SALSA algorithm of the sybil items created by the adversary

D. Giga sybil attack

In a real network, multiple users could potentially mount a sybil attack. Hence, it is worth considering the performance of our network as increasing percentage of all users in the network become malicious users. For the first half of the experiments, we set 50% of all the nodes to sybil nodes and measure the influence gained for sybil songs over sampled users with varying decay parameters like in the previous experiments. This allows us to measure the influence on the score gained from a "giga sybil" attacks with different decay paramaters.

However, after the % sybil users in the network passes a certain threshold, it will be impossible for our system to work effectively since the only items that could be potentially discovered by random walks would be sybil items and hence, due to the influence of beta decays the rankings produced by our system should be effectively random. In our experiments we found this threshold to be around 0.6 for our dataset.

XII. Experiment Results

A. Leave one out cross validation

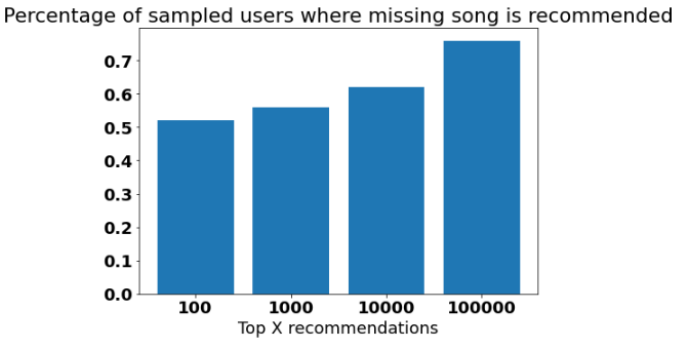


Fig. 8: Leave out one experiment with α = 0.1 and $\beta = 0.0$

As shown in 8 with alpha decay set to 0.1, in more than half of the sampled users, the missing song is within the top 100 recommendations (the dataset consists of 386213 songs so this is top 0.0003% of the results) and about 80% of the times it is in the top 10000 (top 33% recommendations).

Next, we compare our system's performance to a vanilla personalised SALSA recommendation system that doesn't include any of the enhancements mentioned in VIII. Figures 9, 10 and 11 compare the performance of both systems in measuring % of missing songs in top X recommendations. While in the top 1000 and 10000 recommendations our system performs about similar as the vanilla algorithm, in the top 100 recommendations it performs much better. We believe that the top 100 recommendations metric is also the most significant one since users often only care about the very highest recommendations.

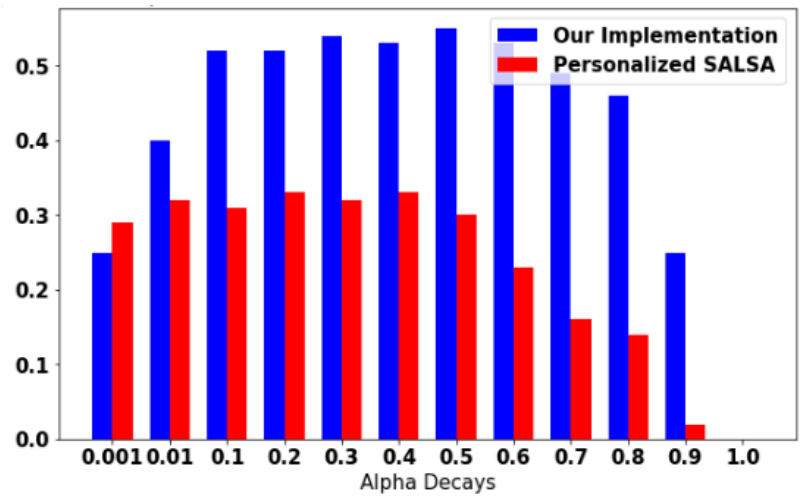


Fig. 9: % of missing songs in top 100 recommendations in both strategies with varying levels of alpha decay

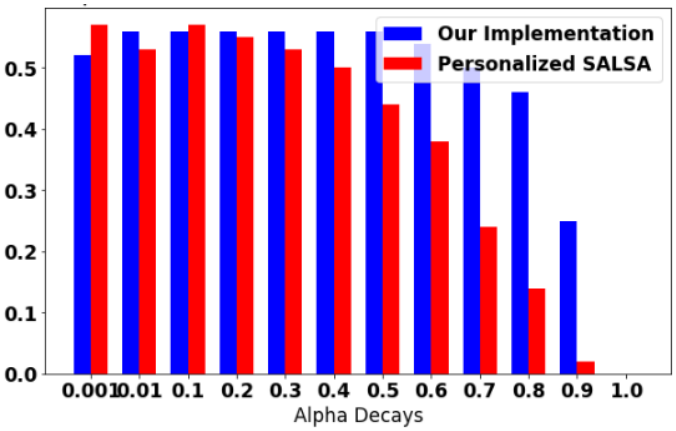


Fig. 10: % of missing songs in top 1000 recommendations in both strategies with varying levels of alpha decay

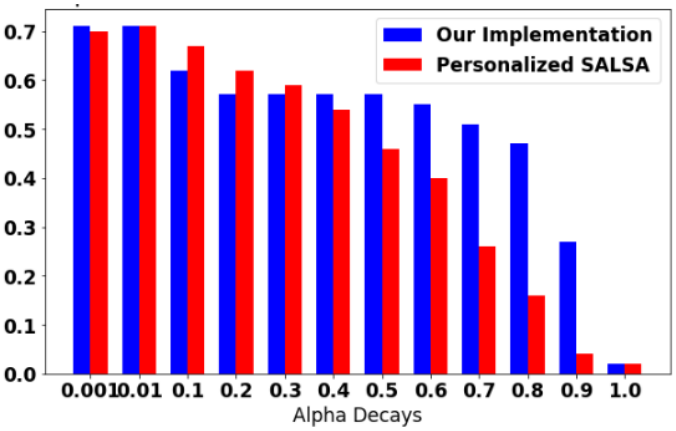


Fig. 11: % of missing songs in top 10000 recommendations in both strategies with varying levels of alpha decay

B. *Effect on ranking of increasing decays*

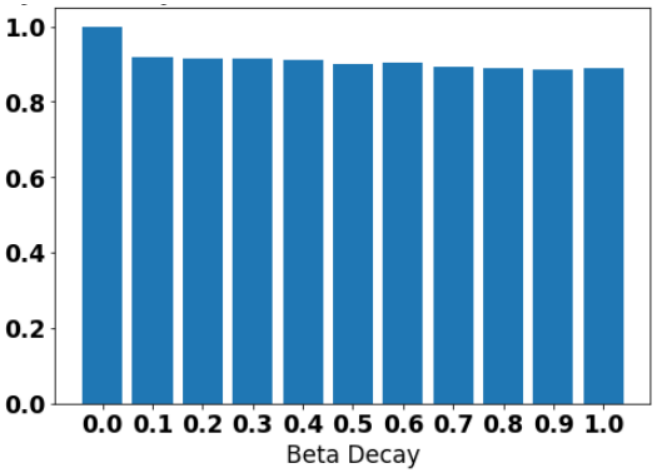


Fig. 12: RBO similarity of non-sybil recommendations with increasing Beta Decay

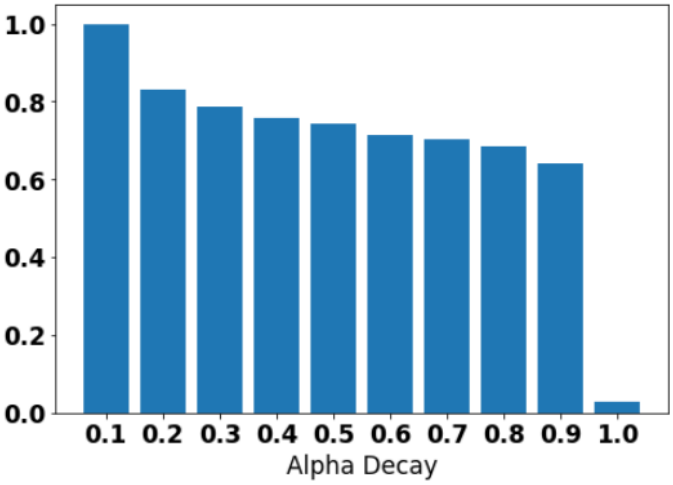


Fig. 13: RBO similarity of non-sybil recommendations with increasing Alpha Decay

As can be seen from figures 12 and 13, the false positive impact seems to increase a bit more with increasing Alpha Decay while increasing Beta Decay seems to have an almost negligible impact. Thus, a real implementation would be better suited to use a low Alpha Decay and a high Beta Decay. In the MusicDAO implementation we set Alpha Decay to 0.1 and Beta Decay to 0.8.

C. *Single sybil attack*

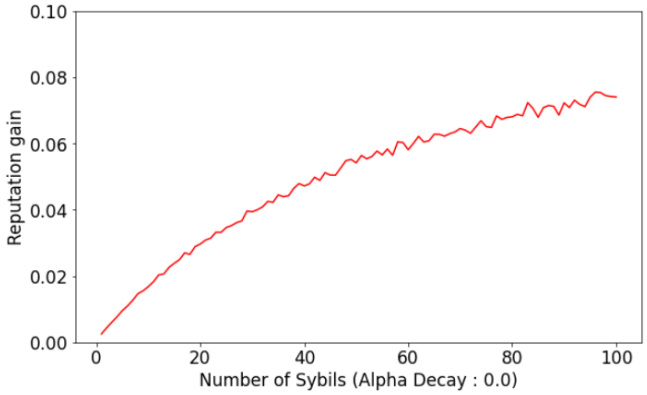


Fig. 14: Ranking score gained for a malicious node with increasingly larger linear sybil attacks (Alpha Decay: 0.0)

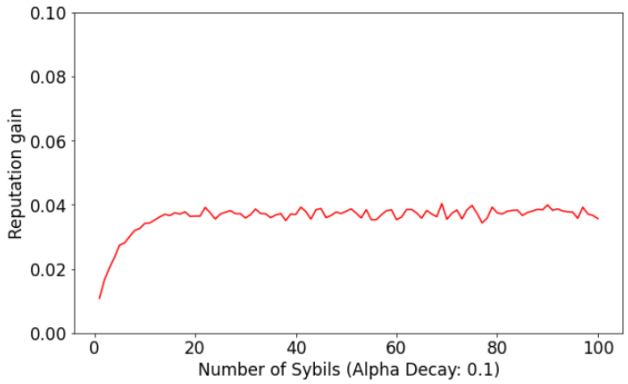


Fig. 15: Ranking score gained for a malicious node with increasingly larger linear sybil attacks (Alpha Decay: 0.1)

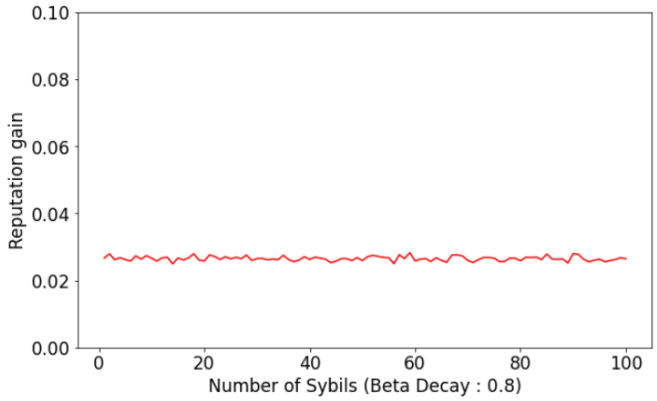


Fig. 16: Ranking score gained for a malicious node with increasingly larger parallel sybil attacks (Beta Decay: 0.8)

First, in 14 we show how an adversary can gain theoretically infinite amount of ranking scores through linear sybil attacks without the presence of Alpha Decay. Then in 15 we rerun the experiment with Alpha Decay set to 0.1 and after a certain number of sybils, the reputation gain for the adversary remains the same, hence creating more sybils doesn't provide any additional benefit to the attacker. In 16 we demonstrate the same effect with Beta Decay set to 0.8 against parallel attacks.

D. Giga sybil attack

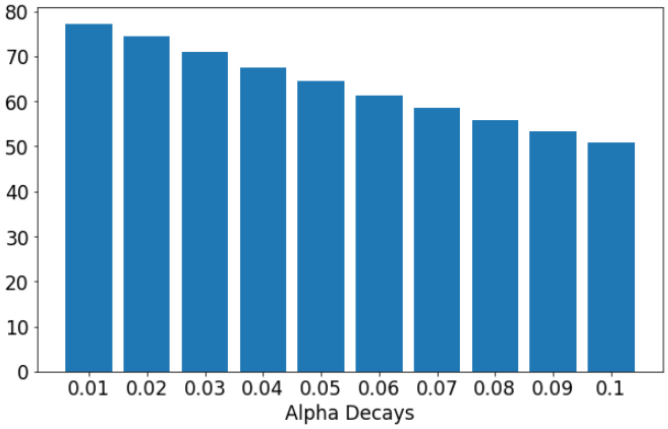


Fig. 17: Cumulative Score for Sybil songs in Giga Sybil Attack with increasing Alpha Decay

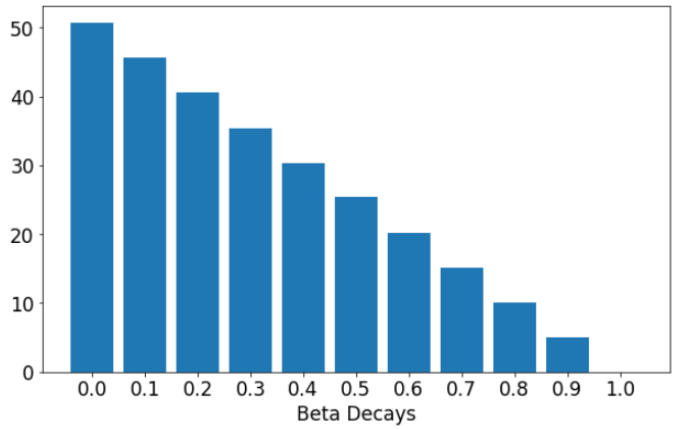


Fig. 18: Cumulative Score for Sybil songs in Giga Sybil Attack with increasing Beta Decay

Figures 17 and 18 show the change in combined ranking score for sybil songs in our Giga Sybil Attack with varying Alpha and Beta decays respectively. Both mechanisms are able to limit sybil gain as expected.

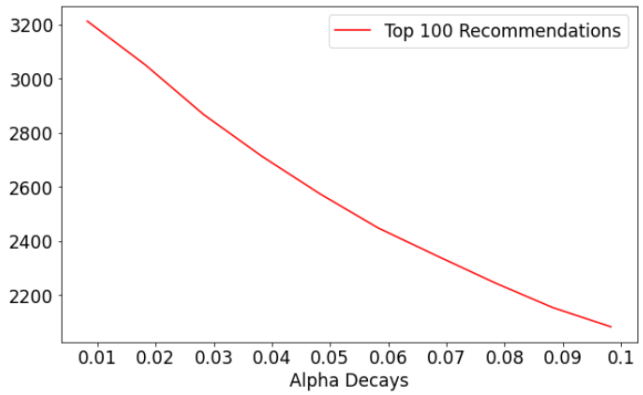


Fig. 19: SITR(100) for Sybil songs in Giga Sybil Attack with increasing Alpha Decay

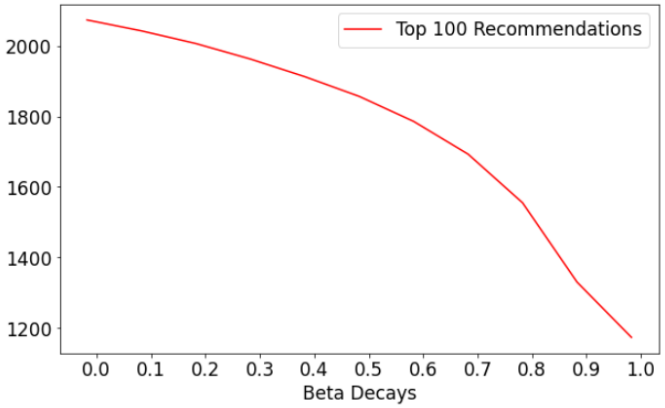


Fig. 20: SITR(100) for Sybil songs in Giga Sybil Attack with increasing Beta Decay (Alpha Decay:0.1)

Figures 19 and 20 show the effect of increasing the decay values on the SITR(100). Note that even though with beta decays we can completely eliminate sybil attack edges, in a giga sybil attack the majority of the items detected are sybil items and hence even though their score is 0, since non-sybil songs were starved of random walks, they have a much lower score too. However, as shown in the graphs, adding the decay values can noticeably reduce the giga sybil attack's gain even if it doesn't eliminate it.

XIII. Testing

All the mentioned components of Web3Recommend's Kotlin implementation are tested using JUnit.

XIV. Real World Deployment

Outside of experiments, as a proof of concept, Web3Recommend was also integrated into MusicDAO, a peer to peer music sharing application that aims to rival Spotify. Figure 1 shows dummy recommendations being generated in the application in a network of 3 users connected together. The code for MusicDAO is open source and can be accessed at https://github.com/Tribler/trustchain-superapp.

XV. Conclusion

In this paper, we presented Web3Recommend, a decentralised Social Recommender System designed to generate trustworthy and relevant recommendations in Web3 platforms on Android. We addressed the challenges posed by decentralised networks, such as the lack of a global perspective and susceptibility to Sybil Attacks.

By integrating the MeritRank decentralised reputation scheme into our graph-based recommendation design, we achieved sybil-resistance in the generated recommendations. Our experiments included evaluations against multiple adversarial strategies. The results demonstrated the trust-relevance balance of our recommendations, showcasing the system's ability to generate personalised, real-time recommendations.

In summary, Web3Recommend represents a significant advancement in the field of social recommender systems. By combining decentralised network principles, the MeritRank reputation scheme, and efficient graph-based algorithms, we have created a sybil-resistant recommendation system capable of generating real-time recommendations in Web3 platforms. Our work not only contributes to the research community but also offers practical benefits to users by enhancing their Web3 platform experiences.

Future research directions could explore further improvements to the decentralised recommendation system, such as enhancing the scalability of the graph-based algorithms, investigating the integration of additional reputation mechanisms, and conducting real-world deployments to evaluate the system's performance and usability in diverse settings. With the increasing adoption of Web3 technologies, the development of trustworthy and relevant recommender systems will continue to play a crucial role in enabling users to discover valuable content and build meaningful connections within decentralised networks.

References

[1] R. Lu, H. Zhu, X. Liu, J. K. Liu, and J. Shao, "Toward efficient and privacy-preserving computing in big data era," *IEEE Network*, vol. 28, no. 4, pp. 46–50, 2014.

[2] F. O. Isinkaye, Y. O. Folajimi, and B. A. Ojokoh, "Recommendation systems: Principles, methods and evaluation," *Egyptian informatics journal*, vol. 16, no. 3, pp. 261–273, 2015.

[3] D. Shenk, "Data smog: Surviving the information glut," 1999.

[4] D. J. Levitin, *The organized mind: Thinking straight in the age of information overload*. Penguin, 2014.

[5] D. Bawden and L. Robinson, "Information overload: An overview," 2020.

[6] P. Hemp, "Death by information overload," *Harvard business review*, vol. 87, pp. 82–9, 121, 10 2009.

[7] Admin, "Tiktok statistics - everything you need to know [mar 2023 update]," Mar 2023. [Online]. Available: https://wallaroomedia.com/blog/social-media/tiktok-statistics/

[8] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh, "Wtf: The who to follow service at twitter," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 505–514.

[9] D. Das, L. Sahoo, and S. Datta, "A survey on recommendation system," *International Journal of Computer Applications*, vol. 160, no. 7, 2017.

[10] I. Guy and D. Carmel, "Social recommender systems," in *Proceedings of the 20th international conference companion on World wide web*, 2011, pp. 283–284.

[11] A. Sharma, J. Jiang, P. Bommannavar, B. Larson, and J. Lin, "Graphjet: Real-time content recommendations at twitter," *Proceedings of the VLDB Endowment*, vol. 9, no. 13, pp. 1281–1292, 2016.

[12] R. Lempel and S. Moran, "Salsa: the stochastic approach for link-structure analysis," *ACM Transactions on Information Systems (TOIS)*, vol. 19, no. 2, pp. 131–160, 2001.

[13] A. Chaabane, Y. Ding, R. Dey, M. A. Kaafar, and K. W. Ross, "A closer look at third-party osn applications: are they leaking your personal information?" in *Passive and Active Measurement: 15th International Conference, PAM 2014, Los Angeles, CA, USA, March 10-11, 2014, Proceedings 15*. Springer, 2014, pp. 235–246.

[14] A. Hassan, "Replication and availability in decentralised online social networks," 2017.

[15] J. Bambacht and J. Pouwelse, "Web3: A decentralized societal infrastructure for identity, trust, money, and data," *arXiv preprint arXiv:2203.00398*, 2022.

[16] B. Nasrulin, G. Ishmaev, and J. Pouwelse, "Meritrank: Sybil tolerant reputation for merit-based tokenomics," in *2022 4th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE, 2022, pp. 95–102.

[17] T. Wissel, "Fairness and freedom for artists: Towards a robot economy for the music industry," 2021.

[18] J. R. Douceur, "The sybil attack," in *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers 1*. Springer, 2002, pp. 251–260.

[19] H. Yu, C. Shi, M. Kaminsky, P. B. Gibbons, and F. Xiao, "Dsybil: Optimal sybil-resistance for recommendation systems," in *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 283–298.

[20] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova, "Monte carlo methods in pagerank computation: When one iteration is sufficient," *SIAM Journal on Numerical Analysis*, vol. 45, no. 2, pp. 890–904, 2007.

[21] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," 2008.

[22] A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl, "Getting to know you: learning new user preferences in recommender systems," in *Proceedings of the 7th international conference on Intelligent user interfaces*, 2002, pp. 127–134.

[23] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[24] A. Borodin, G. O. Roberts, J. S. Rosenthal, and P. Tsaparas, "Finding authorities and hubs from link structures on the world wide web," in *Proceedings of the 10th International Conference on World Wide Web*, ser. WWW '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 415–429. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/371920.372096

[25] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós, "Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments," *Internet Mathematics*, vol. 2, no. 3, pp. 333–358, 2005.

[26] A. Blum, T. H. Chan, and M. R. Rwebangira, "A random-surfer web-graph model," in *2006 Proceedings of the Third Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*. SIAM, 2006, pp. 238–246.

[27] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized pagerank," *arXiv preprint arXiv:1006.2880*, 2010.

[28] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 604–632, 1999.

[29] P. Baran, "On distributed communications networks," *IEEE Transactions on Communications Systems*, vol. 12, no. 1, pp. 1–9, 1964.

[30] G. Korpal and D. Scott, "Decentralization and web3 technologies," 5 2022. [Online]. Available: https://www.techrxiv.org/articles/preprint/Decentralization_and_web3_technologies/19727734

[31] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. van Steen, and H. Sips, "Tribler: a social-based peer-to-peer system," *Concurrency and Computation: Practice and Experience*, vol. 20, pp. 127 – 138, 02 2008.

[32] B. Nasrulin, G. Ishmaev, and J. Pouwelse, "Meritrank: Sybil tolerant reputation for merit-based tokenomics," 2022. [Online]. Available: https://arxiv.org/abs/2207.09950

[33] N. Borisov, "Computational puzzles as sybil defenses," in *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)*. IEEE, 2006, pp. 171–176.

[34] E. Damiani, D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante, "A reputation-based approach for choosing reliable resources in peer-to-peer networks," in *Proceedings of the 9th ACM conference on Computer and communications security*, 2002, pp. 207–216.

[35] H. Rowaihy, W. Enck, P. McDaniel, and T. La Porta, "Limiting sybil attacks in structured p2p networks," in *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 2596–2600.

[36] R. A. Bazzi and G. Konjevod, "On the establishment of distinct identities in overlay networks," in *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, 2005, pp. 312–320.

[37] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman, "Reputation systems," *Communications of the ACM*, vol. 43, no. 12, pp. 45–48, 2000.

[38] P. Resnick, R. Zeckhauser, J. Swanson, and K. Lockwood, "The value of reputation on ebay: A controlled experiment," *Experimental economics*, vol. 9, pp. 79–101, 2006.

[39] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "Sybilguard: defending against sybil attacks via social networks," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, 2006, pp. 267–278.

[40] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, "Sybillimit: A near-optimal social network defense against sybil attacks," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE, 2008, pp. 3–17.

[41] A. Mislove, A. Post, P. Druschel, and P. K. Gummadi, "Ostra: Leveraging trust to thwart unwanted communication." in *Nsdi*, vol. 8, 2008, pp. 15–30.

[42] D. N. Tran, B. Min, J. Li, and L. Subramanian, "Sybil-resilient online content voting." in *NSDI*, vol. 9, no. 1, 2009, pp. 15–28.

[43] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The non-stochastic multiarmed bandit problem," *SIAM journal on computing*, vol. 32, no. 1, pp. 48–77, 2002.

[44] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 661–670.

[45] L. Guo, J. Ma, Z. Chen, and H. Zhong, "Learning to recommend with social contextual information from implicit feedback," *Soft Computing*, vol. 19, pp. 1351–1362, 2015.

[46] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-based systems*, vol. 46, pp. 109–132, 2013.

[47] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Proceedings first international conference on peer-to-peer computing*. IEEE, 2001, pp. 99–100.

[48] D. Michail, J. Kinable, B. Naveh, and J. V. Sichi, "Jgrapht—a java library for graph data structures and algorithms," *ACM Transactions on Mathematical Software (TOMS)*, vol. 46, no. 2, pp. 1–29, 2020.

[49] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. Van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems (TOCS)*, vol. 25, no. 3, pp. 8–es, 2007.

[50] R. Baraglia, P. Dazzi, M. Mordacchini, and L. Ricci, "A peer-to-peer recommender system for self-emerging user communities based on gossip overlays," *Journal of Computer and System Sciences*, vol. 79, no. 2, pp. 291–308, 2013.

[51] A. Crespo and H. Garcia-Molina, "Semantic overlay networks for p2p systems," in *International Workshop on Agents and P2P Computing*. Springer, 2004, pp. 1–13.

[52] Y. Ma, B. Narayanaswamy, H. Lin, and H. Ding, "Temporal-contextual recommendation in real-time," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 2291–2299.

[53] B. Zhang and B. Yuan, "Improved collaborative filtering recommendation algorithm of similarity measure," in *AIP Conference Proceedings*, vol. 1839, no. 1. AIP Publishing LLC, 2017, p. 020167.

[54] I. Cohen, Y. Huang, J. Chen, J. Benesty, J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," *Noise reduction in speech processing*, pp. 1–4, 2009.

[55] D. S. Johnson and M. A. Trick, *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*. American Mathematical Soc., 1996, vol. 26.

[56] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," 2011.

[57] B. McFee, T. Bertin-Mahieux, D. P. Ellis, and G. R. Lanckriet, "The million song dataset challenge," in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12 Companion. New

York, NY, USA: Association for Computing Machinery, 2012, p. 909–916. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/2187980.2188222

[58] W. Webber, A. Moffat, and J. Zobel, "A similarity measure for indefinite rankings," *ACM Transactions on Information Systems (TOIS)*, vol. 28, no. 4, pp. 1–38, 2010.

```
function RandomWalk(currentNode, lastNode, userToUserGraph, userToItemGraph)
    if currentNode is a User then
        WalkToSong(currentNode, lastNode, userToItemGraph)
    else
        WalkToUser(currentNode, lastNode, userToUserGraph, userToItemGraph)
    end if
end function

function WalkToSong(currentNode, lastNode, userToItemGraph)
    userEdgesWeight ← 0
    for all edge e ∈ {edges of currentNode} do
        userEdgesWeight ← userEdgesWeight + userToItemGraph.WeightOf(e)
    end for
    p ← userEdgesWeight * RandomDoubleBetween(0, 1)
    cumulativeP ← 0
    for all edge e ∈ {edges of currentNode} do
        cumulativeP ← cumulativeP + userToItemGraph.WeightOf(e)
        if p ≤ cumulativeP then
            lastNode ← currentNode
            currentNode ← userToItemGraph.GetSongForEdge(e)
            break
        end if
    end for
end function

function WalkToUser(currentNode, lastNode, userToUserGraph, userToItemGraph)
Require: currentNode is an Item and lastNode is User with neighbor currentNode
    lastNodeNeighbors ← userToUserGraph.neighborsOf(lastNode)
    lastNodeNeighborsWithEdgeToCurrentNode ← filterNodesWithMissingEdge(lastNodeNeighbors, currentNode)
    songEdgesWeight ← 0
    for all edge e ∈ {edges of currentNode} do
        if userToItemGraph.GetUserForEdge(e) ∈ lastNodeNeighborsWithEdgeToCurrentNode then
            songEdgesWeight ← songEdgesWeight + userToItemGraph.WeightOf(e)
        end if
    end for
    p ← userEdgesWeight * RandomDoubleBetween(0, 1)
    cumulativeP ← 0
    for all edge e ∈ {edges of currentNode} do
        if userToItemGraph.GetUserForEdge(e) ∈ lastNodeNeighborsWithEdgeToCurrentNode then
            cumulativeP ← cumulativeP + userToItemGraph.WeightOf(e)
            if p ≤ cumulativeP then
                lastNode ← currentNode
                currentNode ← userToItemGraph.GetUserForEdge(e)
                break
            end if
        end if
    end for
end function
```

Fig. 6: Simplified Web2Recommend SALSA Random Walk

Total Users	50000
Total Edges	201114
Max UtU Edges	5
Min UtU Edges	0
Avg UtU Edges	4.02228

a) User to User Graph

Total Users	50000
Total Songs	386213
Total Edges	659962
Max UtS Edges	53
Min UtS Edges	0
Avg UtS Edges	13.19924

b) User to Song Graph

Fig. 7: Test Network Stats (Note that UtU stands for User to User and UtS stands for User to Song)