Unsupervised Continuous Learn-to-Rank for Edge Devices in a P2P Network

Andrew Gold
Delft University of Technology
Delft, The Netherlands
a.w.r.gold@student.tudelft.nl

***Abstract*— Ranking algorithms in traditional search engines are powered by enormous training data sets that are meticulously engineered and curated by a centralized entity. Decentralized peer-to-peer (p2p) networks such as torrenting applications and Web3 protocols deliberately eschew centralized databases and computational architectures when designing services and features. As such, robust search-and-rank algorithms designed for such domains must be engineered specifically for decentralized networks, and must be lightweight enough to operate on consumer-grade personal devices such as a smartphone or laptop computer. This thesis introduces G-Rank, an unsupervised ranking algorithm designed exclusively for decentralized networks. We demonstrate that accurate, relevant ranking results can be achieved in fully decentralized networks without any centralized data aggregation, feature engineering, or model training. Furthermore, we show that such results are obtainable with minimal data preprocessing and computational overhead, and can still return highly relevant local results even when a user's device is disconnected from the network. G-Rank is highly modular in design, is not limited to categorical data, and can be implemented in a variety of domains with minimal modification. The results herein show that unsupervised ranking models designed for decentralized p2p networks are not only viable, but worthy of further research.**

I. INTRODUCTION

The problem of relevance ranking in information retrieval problems has been well-studied for decades, solutions for which have enabled users to query vast swathes of information on the World Wide Web and retrieve highly relevant results within milliseconds. Nascent search-and-rank techniques for web search culminated with PageRank (SOURCE?) in 1998, directly leading to Google's ascendant dominance in the web search domain. All such algorithms, however, depend upon ever-growing databases of mapped relations between various information sources and topics, requiring enormous amounts of computational power to deliver lightning-fast results directly to a user's device. Therefore, these algorithms all depend upon highly centralized information architectures with thousands of skilled attendants dedicated to maintaining and improving system capabilities.

As such, typical ranking algorithms are wholly unsuited for deployment in decentralized information architectures such as peer-to-peer (p2p) file sharing networks (e.g. BitTorrent) and Web3 applications. These networks are overwhelmingly comprised of individual users where the maximum computational and storage capacity available to any search-and-rank algorithm is that of an individual's desktop computer or mobile device. The success of many nascent applications built atop decentralized networks therefore depends upon the efficacy of novel search-and-rank schemes designed specifically for these domains. These algorithms must have a serverless architecture, be lightweight enough to run on a cheap smartphone, and yet be powerful enough to return highly relevant results to each individual user.

Furthermore, these algorithms must adhere to the ethos of these decentralized networks, which often weights user privacy and information security foremost among its tenets. Any ranking algorithm built in such a domain must therefore be able to function effectively utilizing data immediately available to a user of a p2p application, the majority of which is often the user's own data. That is not to say that a ranking algorithm cannot be improved via the sharing of information between participants in such networks, but rather that the algorithm must be entirely self-sufficient and self-contained without any meaningful expectation of obtaining new information outside of the local device. The concept of local-first software is not new (SOURCES), and privacy-preserving machine learning schemes such as encrypted machine learning (SOURCE) and federated machine learning (SOURCES) already exist, yet the problem of storage and overhead persists. Unfortunately, many of these machine learning models are supervised which handicaps developers by requiring large amounts of high-quality training data to achieve meaningful results. Any model that can quickly retrieve relevant information, sufficiently rank the results, and deliver it to the user without the need for training the model before deployment would allow for p2p networks of any size to deliver meaningful search capabilities without needing to bootstrap the model first. Therefore, truly decentralized unsupervised ranking system sits at the forefront of p2p and Web3 communications development.

The rapid growth of p2p file-sharing networks in the early 2000's led to a boom in research for search algorithms designed explicitly for such networks (SOURCES). Many such algorithms attempted to recreate the efficacy of well-known existing search and rank algorithms such as PageRank, yet the number of publications had plateaued and begun to decline around the beginning of the past decade. The explosive growth of blockchain and Web3 technologies has influenced a new generation of developers designing for a more decentralized web experience, yet many search and rank solutions still rely upon trusted sources, application-specific

hardware, or various economic incentives in order to function effectively. Decentralized search and rank algorithms that do not depend upon any centralized entity to function properly, are domain-independent, and can sufficiently replicate the performance of more centralized solutions are still nascent. It is the purpose of this thesis to demonstrate that a simple, lightweight, and effective ranking algorithm can be deployed to p2p applications while achieving respectable results.

TODO: Do we mention Tribler / context at all, or focus purely on p2p file-sharing and streaming networks as a general concept?

This thesis introduces the unsupervised ranking algorithm G-Rank designed explicitly for ranking search results in a p2p torrent-based music streaming platform built atop the Tribler platform. The goal of this first validation experiment is to demonstrate the "correctness" of an unsupervised learn-to-rank (LTR) model in the context of a distributed p2p file sharing network. This model requires no training or bootstrapping to function, is capable of returning relevant results to users within the first few queries, and is not constrained by any dependence upon large datasets. G-Rank is demonstrably capable of ranking results in line with their global popularity, even though the model itself is never aware of any global popularity ranking. G-Rank will quickly approach and oftentimes reach the optimal global ranking, even if a user does not perform any queries themselves; as a network utilizing G-Rank grows in usership, new users will see highly relevant results even with their first query.

The rest of this paper is as follows. Section 2 expounds upon the problem of relevance ranking, namely supervised versus unsupervised methods. Section 3 details the implementation of the G-Rank algorithm, describing the clicklog structure and gossip-based information dissemination mechanism necessary for its functioning, as well as the experimentation and evaluation of the model. Section 4 explores the results of the experiments, comparing them to other ranking algorithms for both traditional as well as decentralized web applications. Section 5 concludes this thesis work.

II. PROBLEM DESCRIPTION

Learning-to-rank is a well-known and thoroughly-studied problem with myriad solutions achieving excellent results, yet many of the most well-known ranking algorithms are designed around centralized data aggregators and supervised training methods. Past research into ranking search results within p2p networks often utilize training, testing, and validation datasets, which besides the traditional pitfalls of supervised learning also constrain the ranking problem into an optimization problem. Compiling relevant datasets and appropriate labels requires considerable effort, which historically has been performed manually by humans and is infeasible for extremely large datasets. Automated labeling methods (such as semi-supervised learning) can speed up this process, but these methods have the drawback of imparting their own inherent bias into the constructed dataset; therefore the difficulty of labeling data in a manual or semi-supervised manner grows faster relative to the increase in size of data.

Other solutions (SOURCES) treat ranking as a recommendation prediction problem, where results are sorted by the predicted score. Framing the ranking problem as a recommendation prediction problem also depends heavily on the manner in which users "score" items that they are recommended. Depending on the application, the manner in which scores are calculated heavily influences the behavior of the recommender. In the domain of e-commerce, an item purchased by a user may be assigned a higher score than an item said user has viewed multiple times but not purchased, even if the user feels that the viewed item is more relevant to them. Meanwhile, a music recommender may assign a higher score to a song that appears in multiple playlists of a specific user yet has fewer overall streaming plays than a song that does not appear in any playlist yet contains a significant number of streaming plays for that same user. As such, the scoring system must be thoughtfully designed for the specific recommendation algorithm and its domain.

With regards to distributed machine learning, federated machine learning has several drawbacks in this domain as well. Federated models are often less accurate due to their relative inability to capture the variance in the overall data throughout the network, as each model is iteratively fitted to a small subset of data. In many such settings, the parameters of the model (e.g. the weights and biases of a trained neural network) are often passed via messages between nodes in the network, training on the local data before passing the model parameters to the next node for further training. This parameter-passing mechanism is often considered sufficient enough to obfuscate local data - affording some degree of user privacy - though recent research has shown that such methods are insufficient to prevent advanced adversaries from identifying users (SOURCES). That being said, any such supervised methods still face the issue of requiring training datasets which limits the scope of potential research due to inadequate training data availability and the infeasibility of synthesizing such datasets oneself. As such, unsupervised ranking algorithms that can approach the performance of supervised ranking methods are much better suited towards p2p domains, where a significant portion of software is open-source and user privacy is often given higher priority than for traditional web services. Significantly reduced overhead in algorithm implementation and maintenance, therefore, is a major benefit for p2p environments. Creating a search engine for a p2p domain that requires no training yet can converge towards an optimal ranking as if an error rate is being minimized in a supervised model would constitute a major development in p2p applications.

III. ARCHITECTURE OF G-RANK

The domain for this thesis is a simulation of a hypothetical music sharing and streaming application built upon a torrent-based p2p network such as BitTorrent. This music application allows users (A.K.A. "nodes" when referring to network architecture) to search other nodes' local libraries and download files to their device. Whenever a user issues a query, the user device appends the query and its associated results to

a clicklog that is stored locally on the device. At various intervals, each device shares a portion of its local clicklog with a selection of other devices in the network via a gossip protocol (See Section 3B). When a user's device receives a gossip message containing updated clicklog information, the device appends the new information to the local clicklog to be used by the ranking model in future queries.

The unsupervised method detailed herein focuses on ranking query search results relative to one another, i.e. pairwise comparison across all potential results. Due to the fact that each node in the network contains only a small subset of total possible search results, it is highly unlikely that any one node attain perfect ranking results without the dissemination of local clicklog information to other nodes in the network. Such a mechanism - be it via message-passing (gossip), broadcasting search history, or a centralized information aggregation scheme - directly and heavily influences the behavior of the unsupervised ranking model. Therefore, the ranking model's co-dependence upon the clicklog dissemination scheme is closely investigated alongside the actual performance of the ranking model, where various dissemination patterns and parameters are considered alongside ranking model parameters and functionality.

A. Unsupervised Ranking Model

When a user searches for a query term, the ultimate goal is to provide the most accurate list of results ranked by relevance to the query term as well as the user. First, the model checks the local clicklog for previous queries of a query term, and if this term has never been queried before it then searches for matches of this term in the metadata of local files stored in the music app, which includes title, artist, and genre tags. The model does not consider misspellings/typos, although methods such as Cubit [2] are highly effective at correcting for typos in information retrieval (IR) schemes and could hypothetically be integrated with G-Rank. If the query term has been seen before, it returns the most popular results for this query weighted by the similarity of search and click behavior of other users who have also issued similar or identical queries (as described in Section 3D).

Due to the fact that the search mechanism considers only the clicklog and item metadata, it is extremely unlikely that a item should erroneously become popularly associated with a query term that has no direct match with any of the item's metadata. The only situation in which this could arise is if a query term has never been seen before nor is contained in any accessible metadata. Should this happen, the search engine returns a list of popular items that have appeared recently in the clicklog. However, because users gossip their local clicklogs in intervals to subsets of other users, it is entirely plausible that a node or subset of the network could be unaware of newly added items with matching metadata at the time of the query. If this were to occur, a user could click on a recommended item that contains no matching metadata to the search query, and then gossip their clicklog history to nearby nodes, who then also perform a search for the same term and click on the same result. Such an occurrence would then erroneously lead to a term-item pairing for which the associated item actually contains no matching metadata, which could then propagate throughout the network.

In order to avoid this situation, search results that contain matching metadata are always ranked above items that have term-item matches in the clicklog yet contain no matching metadata. The justification for such is that should users wish to find a specific item, they ostensibly are aware of the title, artist, album, or some other trait that would be found within the item's metadata such that they need not rely entirely upon the search history of a specific term in order to find said item. A positive side-effect of this restriction is that it also diminishes the effect of adversarial users "query-bombing" the network to negatively influence the performance of the ranking model.

B. Clicklog Structure

Each node in the network contains a clicklog that stores the following information as a row entry: the node's unique ID, the query term, the query results in descending order, the item the user clicked on, the title of the item clicked upon, the tag metadata associated with that item, and a unique key associated with the query term consisting of the concatenation of the node's unique ID and the local query number. The key is used to determine the order of the clicklog without the need for timestamps, as timekeeping in a distributed system is a complex problem that is beyond the scope of this thesis. When a query is performed, the results are stored locally in memory until a user clicks upon a result, after which the clicklog entry is created and appended locally. Whenever a gossip round occurs, only the clicklog entries that have been appended since the last gossip round are included, which dramatically reduces the size of the gossip message[1] as the clicklog grows large. However, as the subset of target nodes included in a gossip round increases, the message space grows superlinearly as each gossip round contains an increasingly large number of duplicate unique queries shared between nodes, even when dropping internal duplicates. On the other hand, as the number of queries performed between gossip rounds increases, the message space grows linearly.

Over time, each node becomes increasingly aware of the click behavior of other nodes in the network without necessarily gleaning insight into the local libraries of said nodes. As such, the dissemination of clicklog data enables the unsupervised model to learn from the behavior of other users without revealing personally identifiable information (PII).

C. Gossiping Clicklogs

The design of the gossip protocol that propagates clicklog information directly affects the performance of the ranking model, and therefore needs to be deliberately designed such that clicklog information is adequately disseminated without congesting the network. In order to determine exactly how

[1]Each query results in approximately 600 bytes of clicklog data.

the gossip parameters affect the model, specific evaluation metrics need to be determined. For example, should a node receive $|K| = 10$ results for a specific query, it is important to determine how many of these results are in the "optimal" ranking, i.e. for each result $k_i \in K$ the distance between the local rank $L(k_i)$ in the above query versus the global average rank $G(k_i)$ across all participants in the network for that query term. In this situation, an item with an "optimal" ranking has a distance of $G(k_i) - L(k_i) = 0$ for any specific query.

In distributed and decentralized networks it is well-understood that obtaining a global "snapshot" of the current network state ranges from "trivial" for small networks to "intractible" for large global networks. Well-known algorithms such as Chandy-Lamport (SOURCE) are still imperfect as they fail to capture incipient changes to the network state deriving from messages that are currently underway during the time of the snapshot, such that by the time the algorithm terminates the state of the network may have already changed. As such, determining a global truth for a p2p network such as the one described in this thesis can only be easily performed in a contained simulation environment in which a global observer aggregates all changes to the network's state. Therefore, it must be understood that any comparisons against a "global" optimum in this thesis come with the caveat that in a live network the global optimum may not be feasibly observable.

D. User Similarity Weighting

As nodes begin to receive new gossip messages, a brief pipeline extrapolates certain patterns from incoming data. After a gossip message is received by node n_i, it appends the new data to its existing clicklog and subsequently searches the incoming data for previously unseen unique node IDs. These unseen node IDs are added to a local list of known nodes, which are then sorted in descending order by the Jaccard similarity between their two clicklogs. The similarity score S between a pair of nodes n_i and n_j is calculated as follows:

- Find the intersection of query terms in the clicklog between n_i and n_j.
- Within this intersection, find all matching click results.
- For each node pair, the number of identical query-click pairs is divided by the overall number of query matches, resulting in a similarity ratio between 0.0 and 1.0, where 1.0 indicates that two nodes have clicked on the exact same item for every single matching query.

That is,

$$S = \frac{|C_i(Q) \cap C_j(Q)|}{|C_i(Q) \cup C_j(Q)|}$$

where $C_i(Q)$ indicates the set of items node n_i has clicked on for query Q. Therefore, the similarity is a ratio of identical query-click results to the overall number of queries shared between two nodes. As such, every node maintains a list of nodes it has become aware of via the clicklog, and determines its similarity to other nodes based on past click behavior. This similarity is then used to weight the results of future queries based on the click behavior of other users, such that users are more likely to see results other similar users have clicked on for similar query terms. Similarity weighting is calculated by taking the dot product between the aforementioned similarity scores for each node and sorted results based on the overall number of clicks found in each node's local clicklog. The resulting ranking R provided to querying node n_i for query Q is therefore calculated as:

$$R(Q) = \forall k \in K_Q, \sum_{i=0}^{N} (C_k \cdot S_i)$$

where S_i indicates the similarity score coefficient for each node $n_i \in N$, and C_k indicates the number of clicks associated with item $k \in K_Q$ where K_Q is the unsorted set of results for query Q. The resulting items are sorted in descending order by their associated scores. As such, each potential query result is assigned a score based on the number of clicks found in each node's clicklog, weighted by the similarity of each node to the node performing the query. Therefore, the dissemination of clicklog data not only informs other nodes of the popularity of items, it also allows for nodes to cluster themselves based on an easy-to-compute metric, further allowing for personalization of results.

A drawback of introducing such a similarity metric into the ranked results is that it introduces a potential attack vector for adversaries to influence the results of future queries throughout the network, e.g. via spam or sybil attacks. Spam attacks become less viable as the number of legitimate users grows larger, while more targeted attacks may be thwarted by the user similarity scheme itself. An adversary attempting to undermine the ranking algorithm by intentionally selecting irrelevant results for specific queries would find themselves increasingly isolated from other users performing legitimate queries, as their behavior over time would continue to deviate from that of other users. Sophisticated adversaries would then need to mimic legitimate behavior for a large portion of their queries in order to remain relevant to other users without ostracizing themselves.

E. Click Modeling

TODO: Discuss how we're clicking?

Modeling realistic user-clicking behavior is essential to the development of ranking algorithms. Not all user clicks may be on relevant items in a list, and as such it can be expected that a certain degree of noise exists in user click data. Extrapolating such noise into a domain such as this thesis therefore requires careful consideration. Without anticipating and modeling a certain degree of noise, a ranking model's query results may erroneously converge towards irrelevant items. There exists a number of methods for modeling real-life user click behavior (SOURCES), however for this thesis users select the highest-ranking item in most queries, except when multiple results with equal relevance scores were shown to the user. In this case, the result with the lowest ID was chosen as a tiebreaker.

IV. Experimentation

The dataset utilized in this thesis was compiled from a series of 256 musical releases by real artists via the PandaCD record label, all of which were released under the Creative Commons license. Entries may be singles, albums, EPs (extended-play releases), and LPs (limited-play releases). Each data entry contains a number of associated "tags" as metadata, which describe the release in terms of genre. These tags have been compiled into a corpus of potential query terms, and every query term in this experiment consists of exactly one tag, of which there are a total of 39 unique tags. Each query term in these experiments is chosen uniformly at random. The simulated network consists of 100 nodes, all of which begin with a limited number of library items. The simulation is initialized as follows:

For each node $n_i \in N, i = \{0, ..., 99\}$ in the network, n_i is initialized without any awareness of other nodes in the network. Upon initialization, a node selects at uniform random ten percent of the items from the global dataset to add to its local library (approximately 26 songs per node). Then, each node n_i adds the clicklog contents of node n_{i-1} to its own local clicklog, i.e. node n_{99} contains a clicklog of 99 items upon instantiation. Next, an initial search is performed. Each node randomly selects a single query term from the corpus and chooses at random one item from its local library with a tag matching the query term, appending this entry to its local clicklog. At this point, no ranking or click modeling is performed for selection.

After every node has been initialized, nodes are chosen uniformly at random alongside a random query term from the corpus to perform a query-term search. The results of the search are ranked as detailed in Section 2A, and an item to be clicked upon is chosen based on the aforementioned click model. The search and click results are appended to this node's local clicklog, and another node is chosen again at random to perform a new search. Every 100 queries, a random subset of nodes is chosen to gossip their clicklog with a randomly chosen subset of other nodes. The number of gossiping nodes, as well as the number of gossip targets, depends **(TODO: Are we doing various gossip sizes?)**

A. Adversarial Simulation

This thesis simulates several adversarial conditions alongside a "baseline" simulation with no adversaries. Each adversarial simulation is intended to isolate and investigate the effects of specific adversarial and anti-social behavior on G-Rank's performance. Each simulation's results is compared to the baseline global performance of G-Rank, as the global optimal rankings are negatively affected by such attacks. As such, each scenario's impact on G-Rank's ability to converge towards a true global optimality without adversarial interference is investigated with the aid of the metrics described in Section 4B.

1) Random Sybil Attack (better name?): The first adversarial simulation implements a basic sybil attack where every five gossip rounds, five new sybil nodes are introduced to the network with no bootstrapping. These sybil nodes perform 50 queries of terms randomly drawn from the corpus, randomly choosing an item in the list of ranked results in an attempt to usurp the current popular rankings for each selected query term. After each sybil node has performed 50 queries, it broadcasts its entire clicklog to the entire network where other nodes append the incoming clicklog to their own local clicklogs. In this scenario, the rate of growth of each node's local clicklog changes from logarithmic to quadratic, as the rate of growth is directly proportional to the number of sybil nodes and random queries performed by the attackers.

2) Targeted Sybil Attack (better name?): The second adversarial simulation is a modification of the previous: instead of choosing a term at random, each sybil attacker chooses a single specific term to perform all 50 queries, clicking the *bottom-most* item in the list of ranked results. After the series of queries are complete, the attackers also broadcast their entire clicklog to the entire network as if it were legitimate clicklog gossip. This attack artificially inflates the relevance score of otherwise low-ranked results, undermining the veracity of the rankings other nodes are shown.

3) Clicklog Inflation: The third adversarial simulation differs from the previous two in that it does not broadcast its clicklog to the entire network. Instead, an adversary performs a significant number of queries in between gossip rounds hoping to be chosen in the next gossip round. If chosen, the adversary will gossip a its large clicklog with its intended targets, resulting in a dramatic inflation in size of non-adversarial node clicklogs. The impact such random gossip is investigated against the more targeted sybil attacks mentioned previously. In large decentralized p2p networks it is highly unlikely that any single node is aware of all other nodes in the network, and furthermore the dissemination of "infected" gossip messages can be mitigated by preventing gossip propagation past an initial recipient. Furthermore, a sufficient gossip protocol can mitigate such threats by randomly choosing a new subset of nodes for each gossip round, relegating the adversary's impact to that of pure chance. Without extended clicklog propagation and deterministic gossip protocols, the impact of clicklog inflation is likely to be dramatically mitigated.

TODO: this sounds more like discussion than explanation, probably not the right section to discuss "potential" impact before the actual results are discussed.

B. Evaluation Metrics

(TODO: local clicklog growth rate, global network bandwidth usage, rate of convergence towards optimality between each gossip round, and...)

V. Results and Discussion

The results of this initial validation thesis experiment show that after just a few gossip rounds, user queries return ranked results that approach or even match the global optimum ranking for each query term. Figure 1 shows that the percentage of queries containing the most popular song per tag grows logarithmically, indicating that early gossip

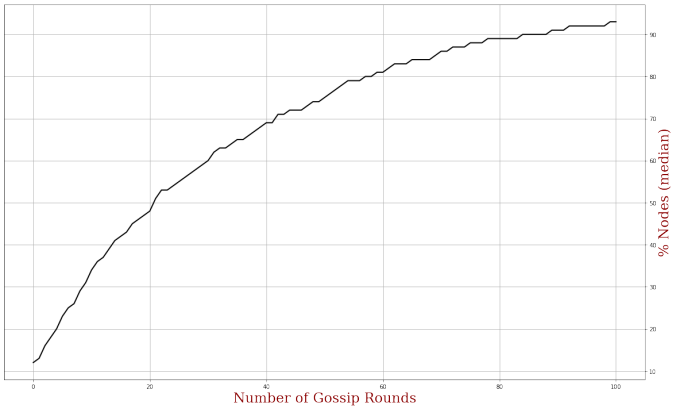

Fig. 1. The median percentage of queries containing the most popular song in the network associated with each possible query term grows logarithmically as the number of gossip rounds increases.

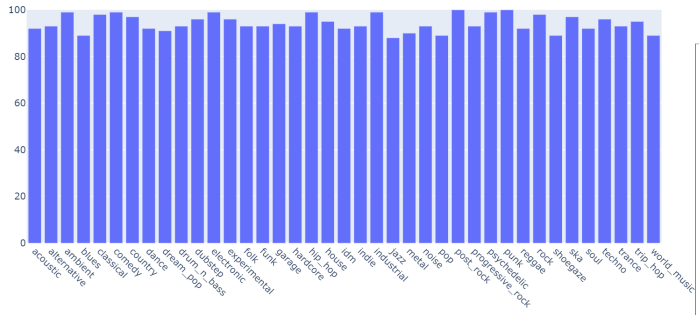

Fig. 2. The percentage of queries containing the most popular song in the network associated with each tag (after 100 gossip rounds).

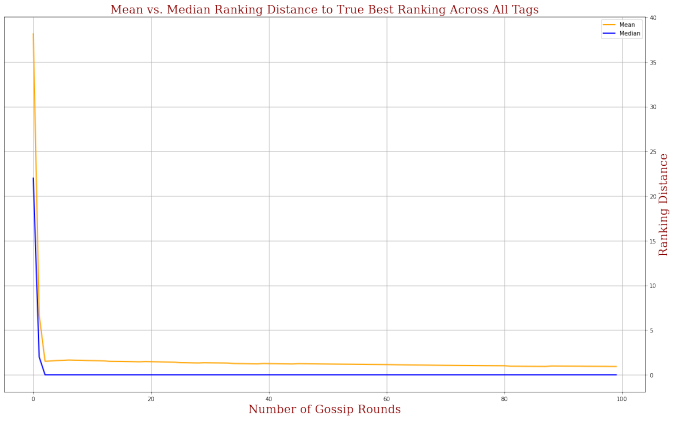

Fig. 3. The mean and median distances between local rankings and the globally optimal ranking across all terms decreases dramatically after just a few gossip rounds, eventually stabilizing at near-perfect optimality for all nodes in the network.

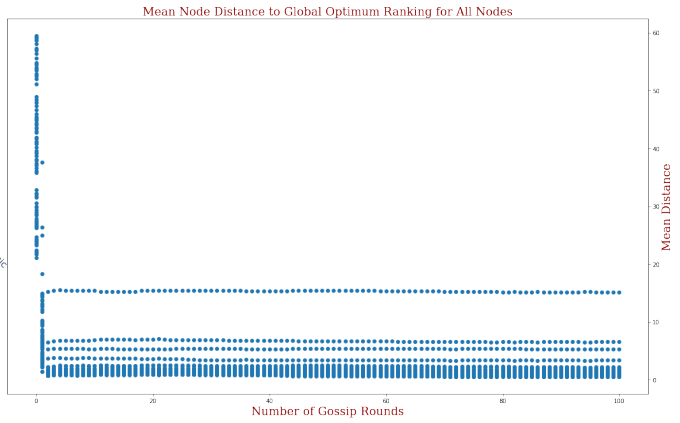

Fig. 4. The average distance of each node's total rankings to the globally optimal ranking across all terms is shown here, where a small number of visible outliers sit well outside the median distance most other nodes converge towards.

rounds have a profound effect on improving search results. As more gossip occurs, the number of queries containing the top song associated with each query approaches 100%, as seen in Figure 2. Figure 2 also shows how the number of most popular songs associated with each possible query term grows at approximately even rates, indicating that the gossip scheme itself does not lead to an imbalance in query term searches.

Furthermore, Figure 3 shows that the distance between each node's local ranking of results and the globally optimal ranking for each possible query term drops dramatically after just a few gossip rounds. Figure 4 expounds upon Figure 3, showing that the vast majority of nodes converge close to the optimal ranking across all tags, with a single notable outlier that does not converge. **(INCLUDE?) Results of the experiment showed that this outlier was a node that received only one round of gossip throughout the simulation, which further demonstrates the impact the gossip of clicklog data has on the unsupervised model.**

A. Adversarial Simulation Results

Considering the known threat that sybil and spam attacks pose to p2p networks, the results of the adversarial simulations generally fall in line with expectations. G-Rank is susceptible to sybil and spam attacks, particularly those with high frequency, though its ability to recover in between attacks and continue converging towards global optimality is notable. When exposed to consistent and persistent spam via both random and targeted sybil attack, G-Rank's ability to recommend near-optimally ranked results suffers greatly. However, clicklog inflation attacks have a diminished effect, impacting only a subset of network participants, likely due to their dependence upon being selected in gossip rounds in order to disseminate "corrupted" clicklogs. This indicates that mechanisms that mitigate or prevent the propagation of messages throughout the network outside of designated gossip rounds may greatly diminish negative consequences of spam and sybil attacks.

As seen in **(TODO: add figure with intermittent spam rounds demonstrating recovery)**, when both random and targeted sybil attacks occur intermittently, G-Rank is able to begin re-converging towards prior benchmarks in ranking. **(TODO: figure with high frequency sybil attacks)** Figure X shows that high frequency attacks result in dramatically diminished ranking performance. However, Figure X+1 shows that high frequency attacks diminish in efficacy if they begin

at later stages of the simulation. This suggests that the length and diversity of each individual node's clicklog has an effect on an adversary's ability to sabotage G-Rank. As the number of gossip rounds increases, the average local clicklog length also grows, affording nodes the opportunity to calculate similarity metrics with an increasingly large subset of other nodes. As the list of known nodes increases, similarity between two distinct clicklogs begins to crystallize such that spam and sybil attackers broadcasting statistically anomalous clicklog data may begin to drift further from other benign nodes.

TODO: not sure if I want to include this next paragraph or not... The user similarity metric described in Section 3D (include hlink) has the potential to include a normalization constant such that when this constant equals 0, clicklog data from nodes with a similarity score of 0 is consequently assigned a weight of 0. This has the consequence of disqualifying any nodes without at least one matching query-click pair with the querying node, which thereby has the knock-on effect of dramatically reducing the probability that an adversary's behavior influences any ranked results. Any influence an adversary then has on ranking is dependent upon the number of matching query-click pairs, which is heavily skewed towards non-targeted attacks such as the random sybil attack and clicklog spam attacks.

Conversely, when the normalization constant is greater than zero, all clicklog data (including malicious entries) is considered in the ranking process as the weight of each entry will subsequently also be a positive non-zero value. The positive effect this has is greater personalization results for benign queries; clicklog results from nodes with similarity scores of 0 will still have the number of clicks associated with that result considered in the final ranking. The negative effect is that all clicklog entries, including malicious entries, will be considered. Figures X+2 and X+3 demonstrates the difference in effect between two identical simulation environments where the normalization constant is either equal to 0 or 1. As seen in these figures, the normalization constant affords benign queries closer convergence to global optimum at the expense of dramatically reduced resilience in the face of attack.

B. Future Work

Potential for future development of unsupervised decentralized search and ranking models in p2p networks is exceptionally rich. G-Rank demonstrates that a simple unsupervised model can recommend near-perfect results to users in sterile network conditions. One of the primary pitfalls of such unsupervised methods is mitigating the threat adversarial actors such as sybil attackers may have on the model. G-Rank is capable of recovering and improving performance after a singular sybil attack, but cannot adapt under consistent and persistent sybil attacks. As such, mitigating threats above the network and protocol layers at the model level is a rich field for future development.

Potential model improvements may include augmenting the user clustering model beyond a simple similarity metric, such that sybil and other spam attacks become classified as outliers with regards to "typical" user behavior, such that recommendation and ranking scores become based off of behavior of other users within clusters.

VI. CONCLUSION

This thesis demonstrates that unsupervised search-and-rank models designed specifically for p2p applications show merit and are worthy of further research. Relatively basic unsupervised methods such as G-Rank are capable of converging towards a global optimum, even when provided with limited data. G-Rank shows considerable resilience in the fact of intermittent sybil and spam attacks, though when subjected to consistent and persistent adversarial interference, the model suffers considerably. As such, spam and sybil mitigation methods implemented between the network and application layers of p2p networks may dramatically improve performance. Another area worthy of further research is spam mitigation within the unsupervised model itself, particularly with regards to outlier detection in clicklog data.

REFERENCES

[1] K. Mochalski, and H. Schulze. "Ipoque internet study 2008/2009." 2009-04-20. http://www.ipoque.com/resources/internet-studies/internet-study-2008-2009 (2009).

[2] Wong, Bernard, Alex Slivkins, and Emin Gun Sirer. Approximate matching for peer-to-peer overlays with cubit. 2008.

APPENDIX

	album	artist	tags
0	Underground Jazz	Marvu	[electronic, rock, indie]
1	Turning	Is World	[electronic, country, hardcore]
2	...And Stars Collide	...And Stars Collide	[industrial, alternative, house]
3	[sun]	Sadsic	[alternative, acoustic, post_rock]
4	Rome	Boreals	[progressive_rock, comedy, dubstep]
...	...	...	...
248	SHOULD WE MAKE IT XXXTRA CREAMY ??	The Jim Tablowski Experience	[soul, world_music, reggae]
249	A private sea	Commonplaces	[ska, industrial, indie]
250	Good Hearted Woman	Maharajah	[techno, ska, indie]
251	11:11	Big Joe Daddy	[garage, noise, progressive_rock]
252	Converzhe	seedmole	[punk, rock, house]

Fig. 5. Dataset

	key	nodeID	term	seen_before	results_order	item_clicked	item_clicked_title	item_clicked_tags
0	[0, 0]	0	acoustic	0	[9, 10, 13, 25, 26, 30, 38, 43, 57, 91, 103, 1...	10	Everything Was	['alternative' 'progressive_rock' 'acoustic']
1	[1, 0]	1	alternative	0	[5, 10, 12, 31, 41, 45, 55, 56, 59, 63, 67, 68...	205	Poshlost	['alternative' 'industrial' 'pop']
2	[2, 0]	2	ambient	0	[2, 16, 28, 30, 47, 52, 53, 69, 82, 117, 145, ...	47	Bear Baiting	['ambient' 'techno' 'world_music']
3	[3, 0]	3	blues	0	[29, 35, 57, 69, 74, 75, 99, 100, 102, 125, 14...	57	Sledding With Tigers/KIDS. Split	['acoustic' 'house' 'blues']
4	[4, 0]	4	classical	0	[1, 23, 61, 73, 80, 85, 110, 123, 133, 137, 13...	161	Medicine	['ambient' 'metal' 'classical']
...	...	...	...	...	...	...	...	...
141	[5, 1]	5	post_rock	1	[69, 199, 55, 173, 47, 6, 106, 93, 33, 64, 37]	69	Narcotic Lion	['ambient' 'blues' 'post_rock']
142	[7, 4]	7	blues	1	[57, 186, 252, 140, 148, 119, 72, 232, 56, 77,...	57	Sledding With Tigers/KIDS. Split	['acoustic' 'house' 'blues']
143	[7, 5]	7	trip_hop	1	[3, 70, 199, 147, 248, 148, 158, 217, 210, 193...	3	[sun]	['soul' 'rock' 'trip_hop']
144	[8, 5]	8	drum_n_bass	1	[106, 206, 94, 142, 39, 159, 170, 80, 53, 237,...	106	We Swim You Jump	['dance' 'trance' 'drum_n_bass']
145	[35, 4]	35	blues	1	[57, 125, 161, 155, 137, 105, 188, 14, 34, 49,...	57	Sledding With Tigers/KIDS. Split	['acoustic' 'house' 'blues']

Fig. 6. Clicklog