Workplace Recommendation with Temporal Network Objectives

Kiran Tomlinson
Cornell University
Ithaca, NY, USA
kt@cs.cornell.edu

Jennifer Neville
Microsoft
Redmond, WA, USA
jenneville@microsoft.com

Longqi Yang
Microsoft
Redmond, WA, USA
longqi.yang@microsoft.com

Mengting Wan
Microsoft
Redmond, WA, USA
mengting.wan@microsoft.com

Cao Lu
Microsoft
Redmond, WA, USA
cao.lu@microsoft.com

ABSTRACT

Workplace communication software such as Microsoft Teams, Slack, and Google Workspace have become integral to workplace collaboration, especially due to the rise of remote work. By making it easier to access relevant or useful information, recommender systems for these platforms have the potential to improve efficient cross-team information flow through a company's communication network. While there has been some recent work on recommendation approaches that optimize network objectives, these have focused on static graphs. In this work, we focus on optimizing information flow, which is highly temporal and presents a number of novel algorithmic challenges. To overcome these, we develop tractable measures of temporal information flow and design efficient online recommendation algorithms that jointly optimize for relevance and cross-team information flow. We demonstrate the potential for impact of these approaches on a rich multi-modal dataset capturing one month of communication between 180k Microsoft employees through email, chats and posts on Microsoft Teams, and file sharing on SharePoint. We design an offline model-based evaluation pipeline to estimate the effects of recommendations on the temporal communication network. We show that our recommendation algorithms can significantly improve cross-team information flow with only a small decrease in traditional relevance metrics.

CCS CONCEPTS

• **Information systems** → **Social recommendation**; **Recommender systems**; • **Theory of computation** → *Social networks.*

KEYWORDS

recommender systems, workplace communication, temporal networks, information flow

ACM Reference Format:
Kiran Tomlinson, Jennifer Neville, Longqi Yang, Mengting Wan, and Cao Lu. 2023. Workplace Recommendation with Temporal Network Objectives. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599932

1 INTRODUCTION

Social communication software such as Microsoft Teams, Slack, and Google Workspace has become a vital component of workplace communication in the last decade, enabling remote collaboration and knowledge transfer between workers [2, 43, 49]. These platforms have become even more important with the increase in remote work due to the COVID-19 pandemic [53]. However, the large quantity of electronic communication can be overwhelming for workers [44]. Recommender systems are a key tool for managing information overload, helping users filter out irrelevant content [55]. In the context of workplace communication software, recommender systems provide an opportunity not only to help users find relevant information, but also to shape the structure of a company's communication network. By bringing information from different parts of a company to a worker's attention, recommender systems can help ideas and resources spread more quickly and efficiently. Traditionally, recommender systems have been entirely relevance-driven—other objectives have only recently begun to be explored [1]. There has been some work on recommender systems with network objectives (e.g., [17, 42, 47]), but the full potential of these systems has not been realized, especially in the context of workplace communication.

We model multi-platform workplace communication as a temporal network and consider how recommendations on one communication platform can increase global information flow. In particular, we focus on the efficient spread of information between *teams* in a company rather than between all individuals, as within-team communication is likely to already be strong—although the same principles apply to individual-level information flow. We consider recommendations in a post-based conversation platform like Microsoft Teams or Slack, where users make posts within *channels*, which are usually centered around a topic, team, or project. Posts in a channel are only visible to the channel's members, who can interact with the post (e.g., replying). Our goal is to recommend posts from channels a user belongs to in a way that increases the speed and quantity of cross-team information flow, without increasing the communication burden on individuals. This can be achieved by efficiently taking advantage of indirect communication: if A tells B something, B can then relay it to C. Additionally, it is important that recommended posts still be relevant to users, or they might become dissatisfied with the system's recommendations. As such, we jointly optimize traditional relevance metrics and network information flow objectives. The need for post recommendations on a platform like Microsoft Teams arises for two reasons: (1) users often belong to many channels with a large number of posts, more than they can easily pay attention to; and (2) users have a limited amount of time and effort, so any way of making it easier for them to access relevant information provides a benefit, allowing them to focus that effort elsewhere.

In contrast with prior work on recommendation with network objectives [11, 17, 35, 42, 47, 57, 58], our networks and network objectives are strongly temporal: it matters *when* communication takes place as well as who participates. Fleshing out the earlier example, if B replies to an email from A and then later replies to a post written by C, B's reply to C can draw on information learned from A—but B's reply to A can't contain information later learned from C's post. We develop an efficient recommendation algorithm, *TIER*, for optimizing two temporal network objectives that capture the speed and quantity of information flow: one existing measure, *information latency* [26] and one novel measure, *total information.* We also analyze the algorithmic problem of finding the edges that most improve these measures. We show these problems are NP-hard, but that the greedy algorithm provides an approximation, as the objectives are submodular—TIER takes advantage of this to make efficient recommendations.

Estimating the impact a recommender will have on a communication network is a particular challenge. One costly and time-consuming approach would be an organization-level A/B test, where different companies use different recommenders during a test period, after which their communication patterns can be measured. As this is impractical, prior recommendation work with static networks used offline evaluation. The typical approach is to add a fixed number of recommended edges to an existing network and evaluate objectives on the augmented graph [11, 17, 42, 47]. In addition to violating our principle of not increasing users' communication burden, this type of evaluation is ill-suited to temporal recommendations. In the temporal setting, the recommendations users have taken in the past influence the structure of the communication network, which future recommendations need to account for. To address this crucial feedback loop, we design an offline evaluation pipeline for recommendation with temporal network objectives. We use a simple and tunable user modeling approach to understand a range of possible outcomes of our recommendation algorithms.

Using our offline evaluation pipeline, we demonstrate the effectiveness of TIER on a rich dataset comprising one month of communication between 180k full-time employees at Microsoft. Our data spans multiple communication platforms: emails on Outlook, posts and chats on Microsoft Teams, and file sharing on SharePoint. In total, this month of activity results in a communication network with over 100 million edges. We focus on recommending Teams posts (using only non-content features for privacy reasons), with the joint objectives of relevance and improved cross-team information flow. We demonstrate a tradeoff between these two objectives. Traditional relevance-based recommenders unsurprisingly score highly in terms of relevance, achieving mean reciprocal rank [50] (MRR) 0.76–0.80, but do not improve cross-team information flow. Conversely, a recommender only optimizing for network objectives significantly improves information flow, but results in very poor relevance (MRR 0.20). TIER allows us to make a small sacrifice in MRR (3-10%) for a large gain in cross-team information flow (50-85% of the gain from a network-objective-only recommender). This tradeoff can be adjusted to taste with a single parameter. See Figure 1 for a visualization of cross-team post communication in the Microsoft data, both with and without our network-driven recommendations. Network-driven recommendations can increase cross-team communication, enabling information to spread more efficiently though a company.

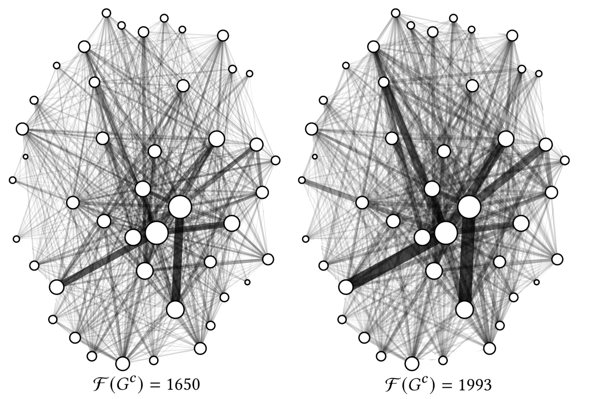

Figure 1: Left: cross-team post communication in Microsoft during March 2022. Right: estimated communication with a network recommender. Edge area is proportional to post interaction count. Total information scores are shown below.

2 RELATED WORK

There has been recent interest in using recommender systems to improve workplace productivity, collaboration, organizational knowledge, and time management [25]. Some prior work has investigated recommender systems for improving knowledge sharing or learning in the workplace [12, 13, 19], although not with temporal communication network objectives. The most closely related line of research to our focuses on recommendation with static network objectives. For instance, there has been work on link recommendations that reduce polarization and insularity in online communities [11, 17, 47, 58], make the distribution of PageRank scores across groups more fair [48], increase the structural diversity of a network [42], increase the influence of specific nodes [57], increase algebraic connectivity [54], and reduce the ability of an adversary to manipulate opinions [3]. In perhaps the closest paper to studying recommendation with temporal network objectives, Parotsidis et al. investigated how to make recommendations that maximally improve a node's expected closeness centrality over probabilistic future graphs [35]. Parotsidis et al. showed their problem to be NP-hard, designed a greedy approximation algorithm, and studied the tradeoff in recommendation accuracy and centrality improvement. Sanz-Cruzado and Castells also designed a recommender system with the goal of improving information spread [42], although they made recommendations in a static network and then ran simulated diffusions for evaluation. In contrast, we explicitly track temporal information flow and make live recommendations according to the current state of a temporal communication network.

There are also several papers that consider non-temporal edge addition problems. For instance, adding k shortcut edges to minimize weighted all-pairs shortest path sums is known to be NP-hard [31], as is minimizing a graph's diameter by adding k shortcut edges [8]. Another related line of work has developed measures of information flow in temporal networks [16, 26, 33, 45].

In another direction, there is a large body of research on recommending a sequence of items over time, for instance in music streaming (called sequential or temporal recommendation) [9, 18, 22, 29, 52]. While such methods are influenced by temporal patterns in user preferences, they still focus on maximizing user engagement and are not directly applicable to optimizing cross-team information flow. However, such methods could be used as the relevance-based recommender component of TIER.

3 DEFINITIONS

We represent a communication network as a temporal directed graph with departure and arrival times for each edge. This allows us to model asynchronous communication across many platforms such as chats, email, and file sharing. Formally, a *temporal graph* [24] G consists of a set of nodes V and a set of temporal edges E, with $n = |V|$ and $m = |E|$. Nodes in G represent individuals in the communication network. A temporal edge is a 4-tuple (u, v, d, a), where $u \in V$ is the source, $v \in V$ the destination, d the departure time, and $a > d$ the arrival time. For example, this could represent an email sent by u at time d that is later read by v at time a. If edges are weighted, we add a fifth element w to the tuple. A *temporal path* [51] P of length k is a sequence of distinct nodes $v_1, \dots, v_{k+1}$ traversed by edges $e_1, \dots, e_k$ where $e_i = (v_i, v_{i+1}, d_i, a_i)$ for $i = i, \dots, k$ and $d_{i+1} \geq a_i$ for $i = 1, \dots, k-1$. The departure time $d(P)$ of a path is the departure time of its first edge, while the arrival time of a path $a(P)$ is the arrival time of its last edge. Let $\mathcal{P}(u, v)$ denote the set of all temporal paths from u to v in G. A temporal graph is *strongly connected* if for every ordered pair of nodes $u, v \in V$ there exists a path from u to v [34] (i.e., $\mathcal{P}(u, v)$ is nonempty for all $u, v \in V$). We use $G_t = (V, E_t)$ to denote the state of G at time t, where E_t only includes edges with arrival times before t: $E_t = \{(u, v, d, a) \in E \mid a \leq t\}$. The set of paths from u to v in G_t is then denoted $\mathcal{P}_t(u, v)$. A *temporal network metric* is a function $\mathcal{F}(G)$ of a temporal graph that computes how effectively information is transmitted within G (discussed further in Section 5).

We assume the collection of nodes V has a natural clustering structure—for instance, teams in a company. That is, there is a natural partition of V into disjoint sets $S_1, \dots, S_k$. Given G and this partition, we can construct another graph of interest: the *cross-cluster communication network* G^c, a coarsening [20] of G according to $S_1, \dots, S_k$. To form G^c, we collapse every node $v \in S_i$ into a *super-node* s_i [28]. Edges in G between $u \in S_i$ and $v \in S_j$ $(i \neq j)$ become edges in G^c between the super-nodes s_i and s_j, with the same departure and arrival times as the original edge. Within-cluster edges are removed. See Figure 2 (1) and (2) for an example.

In addition to the communication network G, we also have a post-based communication platform represented in G in which we make recommendations. On this platform, a *post* $p = (u, t)$ is written by a user $u \in V$ at a time t and published in a *channel*. Each channel has a set of *members*, a subset of V, and users can be members of many channels. Posts in a channel are only visible to the channel's members. If another user v reads p at time t', this adds the *in-edge* (u, v, t, t') to G, representing the transfer of information from the author to the reader. If v replies to the post at t' and the post author u reads the reply at t'', this adds the *out-edge* (v, u, t', t'') to G.

4 PROBLEM STATEMENT

Our goal is to improve cross-team communication in a company by recommending posts (see Figure 2 for an overview of the problem setting). Formally, given a multi-platform temporal communication network G and a temporal network metric $\mathcal{F}$, we want to improve $\mathcal{F}(G^c)$ by recommending posts to users in V, without compromising traditional recommendation relevance metrics. Formally, let $R_{u,t}$ be the ranking of posts recommended to user u at time t. The candidate set of posts for $R_{u,t}$ consists of recent posts (no older than $t - \delta$) from channels u is a member of. Let $y_{u,t} = 1$ if u takes an action on one of the recommended posts (otherwise 0). If a user takes the recommendation and reads a post, this adds the author-reader in-edge (and out-edge, if the user replies to the post) to G. Let $G^c_{R,t'}$ be the cross-cluster communication network resulting from using recommender R up until time t'. Our goal is to maximize

$$\mathcal{F}(G^c_{R,t'}) + \gamma \sum_{R_{u,t} \mid t < t'} y_{u,t}, \tag{1}$$

where γ controls the importance of recommending relevant posts. We thus aim to place posts that will improve $\mathcal{F}(G^c)$ higher in the ranking, encouraging users to interact with them, and therefore influencing $\mathcal{F}(G^c)$. However, in order to achieve our goal of simultaneously maximizing post relevance (note we use actions as binary indications of relevance), we also want to recommend posts a user is more likely to engage with. We now discuss our temporal network metrics $\mathcal{F}$.

5 MEASURES OF INFORMATION FLOW IN TEMPORAL NETWORKS

We consider two measures of temporal information flow: (1) an existing metric, *information latency* [26], and (2) a new metric we call *total information*. Latency captures the age of the most recent information one node could have about another, an idea adapted from computer networking systems [27, 30]. This is a natural metric that has easy-to-interpret time-based units, but fails to capture several salient aspects of temporal information flow, including the degradation of information along long paths (as in the game of telephone [4, 5]) and the quantity of communication. Our total information metric accounts for both, but at the cost of less intuitive units and parameters that require tuning. For simplicity, we define both information latency and total information with respect to the individual-level graph G. For our recommendations, we will apply these measures to the cross-cluster communication graph G^c.

5.1 Information latency

Information latency was originally defined for temporal networks in which each edge has a single timestamp [26] (i.e., $a = d$), but it is straightforward to extend it to account for separate departure and arrival times. The *view* of u with respect to v at time t, denoted $\text{view}(u, v, t)$, is the latest departure time of a path from u to v that arrives no later than t: $\text{view}(u, v, t) = \max_{P \in \mathcal{P}_t(u,v)} d(P)$.

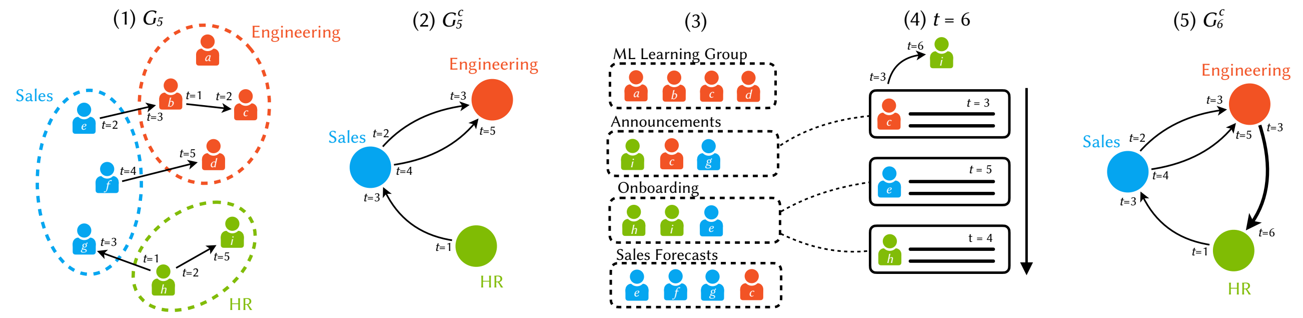

Figure 2: Overview of our problem setting. (1) Temporal communication network G at $t = 5$ with clusters colored. (2) Cross-cluster communication network G^c at $t = 5$. (3) Channel membership on the post platform. (4) Post recommendations for user i at $t = 6$, with dashed lines indicating each post's channel. User i engages with c's post, creating the new communication edge $(c, i, 3, 6)$. (5) Cross-cluster communication network at $t = 6$ with new communication edge.

If no such path P exists, we say $\text{view}(u, v, t) = -\infty$. The *information latency* [26] of a node u with respect to v at time t, denoted $\text{latency}(u, v, t)$, is the shortest amount of time between t and the departure time of a path leaving u and arriving at v no later than t: $\text{latency}(u, v, t) = t - \text{view}(u, v, t)$. Thus, $\text{latency}(u, v, t)$ represents the minimum age of v's knowledge about u: any new idea u has between $\text{view}(u, v, t)$ and t cannot possibly have reached v through G. In order to track all pairwise latencies in a graph, we store them in the latency matrix L_t, where $(L_t)_{uv} = \text{latency}(u, v, t)$. To measure the overall communication in G, we define the latency of G at time t to be the sum of all pairwise latencies: $\text{latency}(G_t) = \sum_{u,v \in V} \text{latency}(u, v, t)$. Note that $\text{latency}(G_t)$ is only finite if G_t is strongly connected. We want to minimize $\text{latency}(G_t)$, so we can think of defining $\mathcal{F}(G_t) = -\text{latency}(G_t)$ in the context of eq. (1).

For latency with a single timestamp per edge ($a = d$), there is a simple single-pass algorithm to compute all pairwise latencies at all times t [26, 27, 30], which can be adapted to our setting with separate departure and arrival times using a priority queue to store in-transit edges. Since the algorithm for latency is similar to the one for total information, we omit a full description.

5.2 Total information

We now turn to our new measure of information flow, *total information*, which differs from information latency in capturing the *quantity* of information flow (direct or indirect) between nodes, in addition to recency. Intuitively, the total information v has about u at time t represents the proportion of u's state that v is aware of at t, under the following assumptions. Nodes always have information 1 about themselves and information 0 about nodes they have never heard from, even indirectly. When a node u communicates with a node v, v learns about the state of every other node u has knowledge of. We assign a weight $w \in [0, 1]$ to each edge e in G representing how efficiently that edge transmits information, from zero transfer ($w = 0$) to perfect transfer ($w = 1$). Additionally, we use exponential decay with rate λ to model the decay of information over time. Let $E_t(v)$ denote the set of all incoming edges to v arriving at time t. We recursively define the total information v has about u at time t as a sum of the information v already had about u at $t - 1$ and the amount of information contained in each new edge. We time-discount the information in each edge, scale by edge weight, and cap total information at 1 at each timestep:

$$\text{totalInf}(u, v, t) = \min\{1, \lambda \,\text{totalInf}(u, v, t - 1) + \sum_{(z,v,d,a,w) \in E_t(v)} w\lambda^{t-d} \,\text{totalInf}(u, z, d)\}. \quad (2)$$

As base cases, we say $\text{totalInf}(u, u, t) = 1$ and $\text{totalInf}(u, v, t_0) = 0$ for times t_0 before the earliest departure in G and all $u \neq v$. As with latency, we store all pairwise total informations in a matrix TI_t where $(TI_t)_{uv} = \text{totalInf}(u, v, t)$. To summarize the overall information quality of a graph, we sum the total information of every pair of nodes: $\text{totalInf}(G_t) = \sum_{u,v \in V} \text{totalInf}(u, v, t)$.

The recursive structure of total information accounts for indirect information flow, while edge weights account for the decay of information over long paths. By summing over incoming edges in total information, we account for the benefit of hearing many times from someone. Our use of a sum corresponds to an assumption that the information contained in different communications is non-overlapping. At the other extreme, we could assume that information has total overlap, and that multiple communications are not beneficial. This can be modeled by taking a maximum rather than a sum, which we call the *maximum information* (defined in Appendix B). Reality is likely somewhere between these extremes, with multiple interactions containing some new and some redundant information. We adopt total information for simplicity and leave the exploration of intermediate metrics to future work.

As with latency, we can compute total information efficiently using a single pass through the edges. The idea is to sweep through edges in temporal order, simulating the spread of information. Paralleling the recursive definition, each edge only requires a local update to total information values, and we can keep track of in-transit edges efficiently using a priority queue. We provide a formal description of this algorithm for single-source information in Algorithm 1, which can be easily extended to all-pairs total information. Given the absence of external knowledge of edge weights w, we calibrate weights so that information values are approximately stable over time. For simplicity, we fix a single weight w_i for each edge type i (email, chat, posts, etc.) See Appendix A for weight calibration details.

Algorithm 1 Single-source total information.

```
1: Input: nodes $V$, temporal edges $E$ sorted by departure time, source node $x \in V$
2: $i_x \leftarrow 1$
3: $i_u \leftarrow 0$, for all $u \neq x$
4: $t_u \leftarrow$ first departure time in $E$, for all $u \in V$
5: $Q \leftarrow$ empty min-priority queue
6: while $|Q| > 0$ or $E$ has unprocessed edges do
7:     if $|Q| = 0$ or the next departure time in $E$ is before the next arrival time in $Q$ then
8:         $(u, v, d, a, w) \leftarrow$ next edge in $E$
9:         if $u \neq x$ then
10:            $i_u \leftarrow i_u \lambda^{d - t_u}$
11:        if $i_u > 0$ and $v \neq x$ then
12:            add $(v, a, i_u w \lambda^{a-d})$ to $Q$ with priority $a$
13:        $t_u \leftarrow d$
14:    else
15:        $(v, a, i) \leftarrow$ next arrival in $Q$
16:        $i_v \leftarrow i_v \lambda^{a - t_v}$
17:        $i_v \leftarrow \min\{i_v + i, 1\}$
18:        $t_v \leftarrow a$
19: $i_u \leftarrow i_u \lambda^{t_{\max} - t_u}$, for all $u \neq x$
20: return $\{(u, i_u) \mid u \in V\}$
```

6 RECOMMENDATION WITH TEMPORAL NETWORK OBJECTIVES

We can now turn to our primary focus: recommending posts that balance cross-team information flow and user engagement, to maximize the objective defined in Equation (1). Our approach, *TIER (Temporal Information and Engagement Recommender)*, balances these objectives by computing relevance and cross-team information scores and ranks posts by a weighted combination of these scores. Weighted score combination is commonly used in multi-objective ranking [32, 56]. The challenging part is computing information scores, as we can simply use the relevance score from any existing relevance-based recommender as an estimate of $y_{u,t}$.

6.1 Optimizing temporal network objectives

In Section 5, we discussed how to compute $\mathcal{F}$ from an observed graph G_t at time t. However, in order to optimize $\mathcal{F}$ for a future time t', we need to select edges for recommendation that will impact the future graph. We will consider the both information latency and total information as our objective $\mathcal{F}$. First, we consider the problem of optimizing information latency. Let $G = (V, E)$ be a strongly connected temporal graph whose last edge arrives at t. We define the following four edge addition problems on G.

Minimum latency in-edges (MLI): given a target node v and a time $t' \leq t$, find k sources $u_1, \dots, u_k$ such that adding edges $\{(u_i, v, t', t'+1) \mid i = 1, \dots, k\}$ minimizes latency(G_{t+1}).

Minimum latency out-edges (MLO): given a source node u and a time $t' \leq t$, find k targets $v_1, \dots, v_k$ such that adding edges $\{(u, v_i, t', t'+1) \mid i = 1, \dots, k\}$ minimizes latency(G_{t+1}).

Myopic minimum latency in-edges (MMLI): given a target node v, find k sources $u_1, \dots, u_k$ such that adding edges $\{(u_i, v, t, t+1) \mid i = 1, \dots, k\}$ minimizes latency(G_{t+1}).

Myopic minimum latency out-edges (MMLO): given a source node u, find k targets $v_1, \dots, v_k$ such that adding edges $\{(u, v_i, t, t+1) \mid i = 1, \dots, k\}$ minimizes latency(G_{t+1}).

Note that for all four of these problems, we can equivalently maximize the view sum instead of minimizing latency. We show that optimizing the first three of these problems is NP-hard. But because the view sum is submodular in the set of added edges, the greedy algorithm that repeatedly picks the edge that most improves latency approximates these hard problems. Due to space constraints, all proofs can be found in an online supplement at https://github.com/tomlinsonk/network-rec-supplement.

THEOREM 6.1. *MLI, MLO, and MMLI are NP-hard.*

THEOREM 6.2. *The greedy algorithm for MLI, MLO, and MMLI* $(1-\frac{1}{e})$*-approximates the optimal final view sum.*

MMLO surprisingly turns out to be easy: the greedy algorithm for selecting edges solves MMLO.

THEOREM 6.3. *The greedy algorithm for MMLO is optimal.*

We can also define the equivalent edge addition problems for total information, in which we maximize totalInf(G_{t+1}) instead of minimizing latency(G_{t+1}). We call these four problems (myopic) maximum total information (in/out)-edges (MTII, MTIO, MMTII, and MMTIO). We show that these problems behave exactly like the latency versions.

THEOREM 6.4. *MTII, MTIO, and MMTII are NP-hard.*

THEOREM 6.5. *The greedy algorithm for MTII, MTIO, and MMTII* $(1 - \frac{1}{e})$*-approximates the optimal final total information.*

THEOREM 6.6. *The greedy algorithm for MMTIO is optimal.*

6.2 TIER

Based on the above theoretical reasoning, we use a greedy approach for recommending posts in order to improve latency and total information. Since we do not know the future edge stream, our approach most closely parallels the myopic edge addition problems. Specifically, suppose we are recommending posts to a user v at time t'. In order to compute the cross-team information score of a post $p = (u, t)$, we measure how much cross-team information flow would immediately improve if recommendation resulted in v reading p or v replying to p. Recall that if v reads p, this would add the in-edge (u, v, t, t') to G and if v replies to p, this would add the out-edge (v, u, t', t'') to G, where t'' is the time when u will receive the reply (since t'' is unknown at recommendation time, an estimate must be used). To measure how much the in- or out-edges would improve cross-team information flow, we maintain the cross-cluster total information and latency matrices TI^c and L^c. Any time cross-cluster communication occurs on any platform we are considering, we update TI^c and L^c according to the corresponding communication edge in G^c. These updates only occur when an edge arrives. As in Algorithm 1, we keep track of in-transit cross-cluster edges in a priority queue along with the source cluster's total information and latency values at time of departure, which is sufficient for computing the update. In other words, we essentially run the all-source version of Algorithm 1 in an online fashion on G^c, processing edges as they occur in the real cross-cluster communication network.

Then, when we want to estimate the potential impact on G^c of recommending a post $p = (u, t)$ to v at time t', we greedily compute the effect that adding the post's in- or out-edge would have on L^c and TI^c. To quantify this effect, we use the total change in latency(G^c) or totalInf(G^c) that would result from this in- or out-edge. We can therefore compute four different network information scores for a post recommendation depending on whether we evaluate impact on information or latency and whether we consider the effect of an in- or out-edge. We call these four scores (and the recommendation algorithms that rank by these scores) *information-in*, *information-out*, *latency-in*, and *latency-out*. Note that if the post author u and the reader v belong to the same team, the post has no effect on cross-team information flow and its network information score is 0.

TIER combines one of these network scores with a relevance score produced by a traditional recommender system. In order to to this, we first normalize both the network and relevance scores to the range $[0, 1]$. Given a collection of posts p to rank, we divide the relevance scores by the maximum relevance score and divide the network score by the maximum network score. Then, given normalized network and relevance scores n_p and r_p for each post p, the TIER score of p is

$$\text{TIER}(p) = n_p + \alpha r_p, \tag{3}$$

where $\alpha \geq 0$ is a tunable parameter controlling the importance of relevance relative to information flow. TIER ranks posts by this combined score, paralleling the weighted objective in Equation (1); in particular, note the equivalence between the parameter α and the weight γ in the objective. Traditional recommender systems can be thought of as using $\alpha = \infty$, while using a smaller α places relatively less importance on relevance. In our experiments, we found that TIER with $\alpha \in [1.5, 5]$ provided significant gains in information flow with only a small decrease in relevance, although appropriate value for a particular application depends on the priorities of the system designers. When we want to emphasize which network or relevance score TIER is using, we append it to the acronym (e.g., TIER-information-out).

6.2.1 Time and space complexity of TIER. Recall that we have k teams. Suppose we need to rank n posts and that the engagement-based recommender takes $f(n)$ time to produce the relevance scores r_p. The running time of TIER for a single recommendation is then $O(f(n) + nk)$—that is, TIER adds $O(k)$ time per post to incorporate its potential impact on cross-team information flow. Since we typically have a small number of teams k, this overhead is also small. In the background, we need to update the $k \times k$ matrix of cross-team information scores (either total information or information latency) as the stream of temporal edges arrives. Given a temporal stream of m edges, of which at most c are in transit at any one time, this takes total time $O(m \log c)$ using a priority queue to store in-transit edges (Algorithm 1). Finally, the additional space cost of using TIER is $O(k^2)$, from the matrix of cross-team information scores. All of these costs are low, allowing TIER to scale to large organizations.

6.3 Offline evaluation

Evaluating network-wide impacts of a recommender system is very challenging; an experimental A/B test would require deploying multiple recommenders to large collection of organizations, each with their own communication network. Due to this difficulty, existing work on recommendation with network objectives has used a variety of offline evaluation approaches, including removing a subset of real edges and then adding back all recommended edges from this subset [35, 54], adding all top-k recommended edges for each user [42, 47], and globally adding a fixed number of edges [11, 17]. These previous approaches are not well-suited to our temporal setting, as it is unrealistic to expect users to increase their quantity of communication. Additionally, our recommendations rely on the current informational state of a network, which is itself influenced by the recommender. This feedback loop is crucial for realistic evaluation of our methods. To see why, suppose a there are two clusters S_1 and S_2 who have high mutual latency or low information about each other. We might therefore want to recommend a post from S_1 to someone in S_2. If the recommendation is taken, then S_2 now has more information about S_1. If we don't account for this in our recommendations, we may keep recommending S_1 posts to S_2 users ad infinitum, at the expense of other cluster pairs.

With this feedback loop in mind, we now provide an overview of our offline evaluation approach and then describe each component in detail. Our historical interaction data is a stream of user actions on a variety of platforms (email, chats, posts, and file sharing), which we convert into temporal edges (u, v, d, a) as described in Section 7.1. In evaluating the recommender R, we keep all of these interactions fixed except on the platform where we make recommendations (posts). Intuitively, our evaluation of R asks what the cross-cluster communication network would have looked like if users sometimes interacted with posts recommended by R instead of the posts they would otherwise have interacted with, keeping the total amount of user activity constant. Specifically, we simluate a cross-cluster communication network G^c_R under recommender R as follows:

(1) Add all edges from platforms we are holding constant (email, chats, file sharing).
(2) On the recommendation platform, for each true action by user u at time t on a post p by author v, we:
   (a) Construct a candidate set of posts for user u at time t,
   (b) Rank the candidates according to R,
   (c) With probability ρ, simulate u acting on a top-k recommended post p' instead of p—otherwise, keep the action on p,
   (d) If u and v are in different clusters, add the edge corresponding to the user-post action (true or recommended) to G^c_R.

More precisely, we process edges in temporal order, allowing G^c_R to populate until the beginning of the test period. During the test period, we simulate a recommendation instance whenever we encounter a post interaction where user u replied or reacted at time t to a post p authored by v no older than 3 days. In each recommendation instance, we construct a candidate set of recent posts to recommend from channels u is active in, downsampling to 100 such posts if necessary and always including the true positive interaction on post p. We rank the candidate posts according to the recommender R and compute accuracy-based ranking metrics according to the rank of the true post p that u interacted with. Finally, we model whether the user takes one of the recommendations (deferring the details of this model for a moment). If the model says no, we keep the true interaction edge and add it to G_R^c. If the model says that u takes the recommendation of a post by v', we instead add an interaction edge between u and v' to G_R^c. When replacing a true action with a simulated one, we keep the interaction type (react or reply) the same as in the true action. Once this procedure is complete, we have a simulated version of what the cross-cluster communication network might look if the recommender R were deployed. We can then evaluate the latency and total information of the simulated network G_R^c in comparison with the true cross-cluster communication network G^c. We don't even need to perform another pass through the edges to do this, as long as we store the state of the cross-cluster total information and latency matrices TI^c and L^c during the first pass—which we need anyway for the network-based recommenders. We take checkpoints of TI^c and L^c once per day in the historical data stream to track the simulated effects of our recommendations over time.

The final component of the evaluation procedure left to describe is the model for whether users take action on the top-k recommendations from R. Learning such a model from historical user-post interaction is extremely difficult, as the recommendations we provide present a distributional shift in the posts users are exposed to. For instance, historically users may have rarely interacted with cross-cluster posts, but this may change when we actually recommend them. To avoid the issues this distribution shift presents, we take a simple approach: we assume a fixed user action probability ρ. That is, we say users take one of the recommended actions with probability ρ and otherwise take the historical action with probability $1-\rho$. We can then adjust ρ to understand a range of possible outcomes of the recommender system. Conditioned on taking a recommended action, we assume users always interact with one of the top-5 recommended posts, sampled with a position bias [7], where higher ranked posts are more likely to be chosen. Specifically, we sample from top-5 posts with weights proportional to $5, 4, 3, 2, 1$ (i.e., the rank-1 post is chosen w.p. $5/15$, the rank-2 post w.p. $4/15$, etc.). Post hoc validation on historical Teams post ranking data found this to be a good match for actual user position bias—albeit under traditional engagement-based recommenders.

7 EXPERIMENTAL EVALUATION

In this section, we describe our real-data experiments. We use one month (March 2022) of communication data among approximately 180k Microsoft employees, including channel posts and chats on Microsoft Teams, emails on Outlook, and file sharing on SharePoint, resulting in a temporal graph with over 100 million communication edges. We then design an offline temporal evaluation procedure for recommendation with temporal network objectives. Finally, we describe the relevance-based recommenders we tested and present the weight calibration approach we used for total information.

7.1 Data description

We use a communication dataset covering one month (March 1, 2022 to April 1, 2022) of interactions between approximately 180,000 full-time employees at Microsoft. We gathered anonymized interactions (using only non-content features for privacy reasons) on Microsoft Teams, Outlook, and SharePoint during this period. In total, these data streams represent over 100 million interactions, an average of ~ 20 outgoing communications per person per day. We now describe how we converted these interactions into a temporal communication graph G and the types of clusters we used to form the cross-cluster graph G^c.

7.1.1 Teams posts. Posts in Microsoft Teams are made in a particular channel and are only visible to members of that channel. Users can *react* to posts with emojis or *reply* to posts with text comments. We collected all Teams posts, replies, and reacts made during the data period, storing anonymized user IDs, timestamps, and anonymized channel IDs. For each react by a user v at time t' on a post written by u at time t, we added a temporal communication edge (u, v, t, t') to G indicating the transfer of information from the post author to the reactor. We did the same for replies, but also added the out-edge (v, u, t', t')[1] to indicate the transfer of information from the replier to the post author. We say that a user is *active* in a channel if they perform at least one post action (react or reply) in that channel any time in our data. In our recommendation task, we only suggest recent posts (no more than δ = three days old) from channels a user is active in. Almost 90% of reacts and replies in our data were on recent posts.

7.1.2 Teams chats. Chats in Teams are group conversations between two or more users. Whenever a user v sent a message to a chat at time t', we assumed that they read every message in the chat since the last time v sent a message to that chat. For each such earlier message written by u at time t, we thus added the edge (u, v, t, t') to the communication network G.

7.1.3 Outlook emails. We gathered timestamped email send and receive actions from the Microsoft Outlook email client. For each email sent by u at time t and received by v at time t', we added the communication edge (u, v, t, t') to G. We filtered out mass emails by ignoring any email received by more than 100 recipients.

7.1.4 SharePoint file actions. Finally, we gathered two types of timestamped file actions from SharePoint: (1) author actions (uploads and edits) and (2) viewer actions (downloads and views). We ignored files that had no author actions during our data period and restricted to user-readable file types (Word, Excel, and PowerPoint documents plus PDFs). Whenever a user v performed a viewing action on a file f at time t', we considered v to have received information from every user who took an author action on f since the last time u took a viewer action f. For each such author action by u at time t, we added the temporal edge (u, v, t, t') to G.

7.1.5 Clusters. We considered three clusterings of Microsoft employees, two based on empirical email patterns and one based on the organizational chart (*org chart*) of the company. For the org chart clusters, we grouped all employees by the subtree they belong to in the organizational hierarchy. We considered subtrees rooted two steps below the CEO, resulting in 79 clusters. For the empirical

[1]Since our data doesn't say when the reply was read, we use the reply time as the edge arrival time.

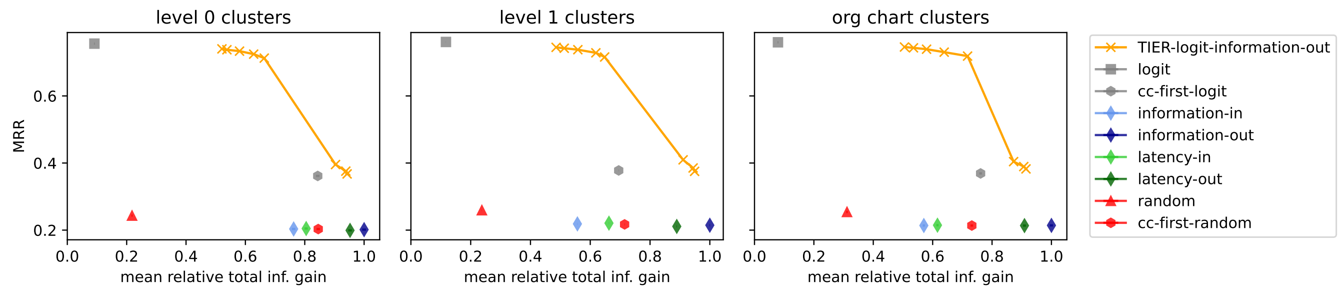

Figure 3: Relevance vs information information flow tradeoff ($\rho = 0.01$). Each point represents a different recommendation algorithm. The x-axis shows the average improvement in cross-cluster total information compared to the true edge stream (higher is better), as a proportion of the gain under *information-out*. Horizontal error bars (tiny) show standard error in information improvement. The y-axis shows each recommender's MRR. For TIER, α ranges from 5 at the top left to 0.5 at the bottom right (all values: 0.5, 0.75, 1, 1.5, 2, 3, 4, 5).

clusters, we used two levels from a hierarchical clustering of employees based on a graph embedding of their email communication network (level 0 = 45 clusters, level 1 = 794 clusters). We ran separate experiments evaluating cross-cluster information flow in G^c for each of these clusterings. For a visual example of a clustering, see Figure 1; this used level 0 clusters (as well as the *information-out* recommender and action probability $\rho = 1$).

7.2 Recommendation algorithms

We compare TIER to four other types of recommenders described below. Since we are evaluating methods along two axes (ranking relevance and resulting information flow), we compare TIER against traditional methods that optimize only for relevance and methods that optimize only for information flow, as well as several baselines.

Relevance only: *logit* and *lightgbm* [23] rank posts by the predicted probability that a user will interact with them using multinomial logistic regression and gradient-boosted decision trees, respectively. We use simple predictive models since they are widely used in industry [6, 46] and are often competitive with much more complex and costly neural models [14, 37]. We use the same set of post features for *logit* and *lightgbm*, including count-based features (e.g., reply and react counts on the post, author post count, channel action and post counts) and temporal features (e.g., post recency, action recency, user-post action recency). We trained both models on the first 30% of post interactions after a burn-in period. See Appendix C for a full list of features and training details.

Information flow only: the *information-in*, *information-out*, *latency-in*, and *latency-out* rankers order posts by their corresponding information flow scores as defined in Section 6.

Random baseline: *random* simply shuffles candidate posts. Note that even random recommendations are a relatively strong baseline, since we only rank "hard negatives" [21, 40] that were recently posted in a channel the user is active in. As such, these are all posts that users could reasonably take interest in.

Cross-cluster first baselines: *cc-first-random*, *cc-first-logit*, and *cc-first-lightgbm* always rank cross-cluster posts higher than within-cluster posts. Among those two categories, they rank posts using *random*, *logit*, or *lightgbm*.

TIER: we tested all eight combinations of the information flow and relevance recommenders: *TIER*-{*information-in*, *information-out*, *latency-in*, *latency-out*}-{*logit*, *lightgbm*}. For each configuration of TIER, we tested relevance weights $\alpha = 0.5, 0.75, 1, 1.5, 2, 3, 4, 5$.

8 RESULTS

We begin by summarizing the average performance of each of the ranking algorithms in terms of relevance and information flow (total information and latency). For information flow measures, this average is taken over daily snapshots and over cluster pairs.

Average performance. In Figure 3, we show the average relevance score and total information gain of different recommendation algorithms using each of the three clusterings and a conservative action probability $\rho = 0.01$. We focus on the logit-based recommenders since they achieved a better tradeoff between relevance and information flow and plot information gain relative to purely optimizing for information flow using *information-out*. See Appendix C for equivalent plots with *lightgbm*, latency, and high action probability $\rho = 1$. As expected, the relevance-based recommenders, *logit* and *lightgbm*, achieve high mean reciprocal rank (MRR): 0.76 and 0.80, respectively. However, they have essentially no effect on information flow compared to the true edge stream. Random recommendations perform much worse in terms of relevance (MRR 0.24-0.26), although they do somewhat improve cross-cluster connectivity. Ranking cross-cluster posts first achieves even better total information, but again poor relevance (MRR 0.39 at best). The best total information is achieved by one of the network recommenders, *information-out*, but it scores even worse in terms of relevance (MRR 0.20-0.21). Similarly, *latency-out* achieves the highest latency improvement overall and very low MRR (see Appendix D). TIER allows for significant gains in information flow: at least 60% as much as *information-out*, and almost as much ranking all cross-cluster posts first. At the same time, TIER still achieves high MRR (0.70 − 0.78 depending on α).

Varying the action probability. In Figure 4, we show how the information flow results are impacted by the simulated user action probability ρ. We tested $\rho = 0.01, 0.05, 0.2, 1$. For clarity, we only plot the best-performing network recommender for each network measure, logit-based relevance recommenders, and TIER-logit with

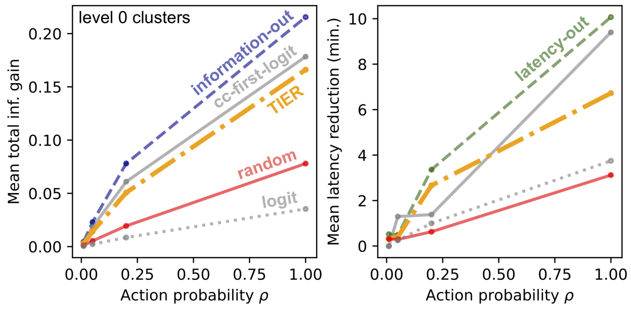

Figure 4: Mean total information gain (left) and latency reduction (right) under different recommenders for different action probabilities ρ (level 0 clusters). The true edge stream has mean total information 0.66 and 47 minute mean latency. TIER uses *logit* and *information-out* (left) or *latency-out* (right), and $\alpha = 2$. The more likely users are to accept recommendations, the more opportunity we have to improve information flow metrics.

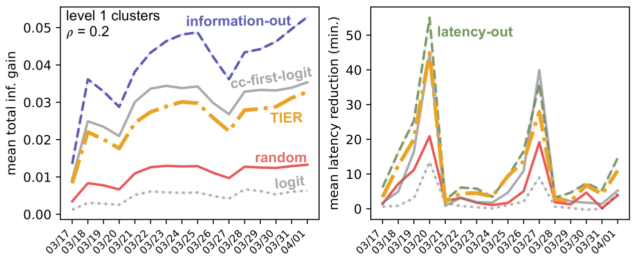

Figure 5: Total information gain and latency reduction trajectories (level 1 clusters, $\rho = 0.2$). Each line shows the improvement in information flow from a recommender over the true edge stream through the test period, averaged over all pairs of clusters. Shaded regions around each line show standard error over cluster pairs. In these plots, TIER uses *logit*, *information-out* (left) or *latency-out* (right), and $\alpha = 2$.

$\alpha = 2$. When users take more of our recommendations (higher ρ), we have more of an opportunity to impact cross-cluster communication. However, there are diminishing returns—notice the change in slope at $\rho = 0.2$. For latency, low action probabilities produce noisy results due to the coarseness of daily checkpointing relative to the small cross-cluster latencies.

Effects over time. In Figure 5, we plot information and latency gain trajectories of different recommenders during the test period for level 1 clusters and action probability $\rho = 0.2$. We plot the same recommenders as in the action probability sweep. With $\alpha = 2$, TIER gets high MRR 0.72-0.76 while providing significant and sustained gains to information and latency (almost as much as *cc-first-logit*). Note that 3/20 and 3/27 were Sundays—cross-cluster latency gains spike on weekends, while total information gains decay over weekends.

Change in cross-cluster edges. There are two ways that our network recommenders increase cross-cluster information flow in our simulations: by replacing intra-cluster communication with cross-cluster communication and by focusing user attention on the the most efficient cross-cluster posts (in terms of information flow). Since we keep the total amount of user activity fixed, any increase in cross-cluster communication must be accompanied by an equal absolute decrease in intra-cluster communication. However, because most communication is intra-cluster, we only need to sacrifice a small *proportion* of intra-cluster edges to achieve larger proportional gains in cross-cluster communication. We illustrate this principle in Table 1, where we show the percent change in intra- and cross-cluster post edges resulting from TIER-logit-information-out (with level 0 clusters and $\alpha = 2$) over the range of action probabilities ρ. The proportional gain in cross-cluster post edges is over 10× the proportional loss in intra-cluster edges.

Table 1: Percent change in post edges from TIER.

	$\rho = 0.01$	$\rho = 0.05$	$\rho = 0.2$	$\rho = 1$
intra-cluster	−0.06%	−0.30%	−1.19%	−5.83%
cross-cluster	+0.63%	+3.16%	+12.48%	+60.89%

9 CONCLUSION

In this work, we took a first step in studying recommendation algorithms with temporal network objectives, focusing on the potential for impact on workplace communication. Specifically, we focused on optimizing cross-team information flow in organizational communication networks. We developed tractable measures of temporal information flow and used these in the design of TIER, an efficient online recommendation algorithm that jointly optimizes for relevance and information propagation. We evaluated our proposed method on a dataset of 100M+ interactions among 180k Microsoft employees and showed that our recommendation algorithms can significantly improve cross-team information flow with only a small decrease in traditional relevance metrics.

Our investigation was necessarily limited in scope; for instance, we assumed a simple model of user-post interaction for our offline evaluation and our total information metric assumes that different incoming communications never overlap in informational content. Our approach to optimizing information flow is also myopic, only considering the impact of the recommended edge. Each of these limitations invites additional exploration in future work. One important direction is the development of additional evaluation mechanisms that can account for downstream network-wide effects such as graph-based off-policy evaluation and A/B testing for information flow. Other future directions include exploring other network metrics, such as combining total and maximum information to account for a mixture of redundant and novel information, and accounting for likely future edges.

REFERENCES

[1] Bushra Alhijawi, Arafat Awajan, and Salam Fraihat. 2022. Survey on the Objectives of Recommender System: Measures, Solutions, Evaluation Methodology, and New Perspectives. *Comput. Surveys* (2022).
[2] Ahsan Ali, Waseem Bahadur, Nan Wang, Adeel Luqman, and Ali Nawaz Khan. 2020. Improving team innovation performance: role of social media and team knowledge management capabilities. *Technology in Society* 61 (2020), 101259.
[3] Victor Amelkin and Ambuj K Singh. 2019. Fighting opinion control in social networks via link recommendation. In *KDD*. 677–685.

[4] Fritz Breithaupt, Binyan Li, Torrin M Liddell, Eleanor B Schille-Hudson, and Sarah Whaley. 2018. Fact vs. affect in the telephone game: All levels of surprise are retold with high accuracy, even independently of facts. *Frontiers in Psychology* 9 (2018), 2210.

[5] Taylor N Carlson. 2018. Modeling political information transmission as a game of telephone. *The Journal of Politics* 80, 1 (2018), 348–352.

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. 7–10.

[7] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. 2008. An experimental comparison of click position-bias models. In *WSDM*. 87–94.

[8] Erik D Demaine and Morteza Zadimoghaddam. 2010. Minimizing the diameter of a network using shortcut edges. In *Scandinavian Workshop on Algorithm Theory*. Springer, 420–431.

[9] Ziwei Fan, Zhiwei Liu, Jiawei Zhang, Yun Xiong, Lei Zheng, and Philip S Yu. 2021. Continuous-time sequential recommendation with temporal graph collaborative transformer. In *CIKM*. 433–442.

[10] Ciprian Florescu and Christian Igel. 2018. Resilient backpropagation (RPROP) for batch-learning in TensorFlow. In *ICLR Workshop Track*.

[11] Kiran Garimella, Gianmarco De Francisci Morales, Aristides Gionis, and Michael Mathioudakis. 2017. Reducing controversy by connecting opposing views. In *WSDM*. 81–90.

[12] Shuang Geng, Lijing Tan, Ben Niu, Yuanyue Feng, and Li Chen. 2019. Knowledge recommendation for workplace learning: a system design and evaluation perspective. *Internet Research* (2019).

[13] Natalie Glance, Damián Arregui, and Manfred Dardenne. 1999. Making recommender systems work for organizations. In *PAAM*, Vol. 99. Citeseer, 19–21.

[14] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems* 34 (2021), 18932–18943.

[15] Mark S Granovetter. 1973. The strength of weak ties. *Amer. J. Sociology* 78, 6 (1973), 1360–1380.

[16] Peter Grindrod, Mark C Parsons, Desmond J Higham, and Ernesto Estrada. 2011. Communicability across evolving networks. *Physical Review E* 83, 4 (2011), 046120.

[17] Shahrzad Haddadan, Cristina Menghini, Matteo Riondato, and Eli Upfal. 2021. RePBubLik: Reducing polarized bubble radius with link insertions. In *WSDM*. 139–147.

[18] Casper Hansen, Christian Hansen, Lucas Maystre, Rishabh Mehrotra, Brian Brost, Federico Tomasi, and Mounia Lalmas. 2020. Contextual and sequential user embeddings for large-scale music recommendation. In *RecSys*. 53–62.

[19] Yvonne M Hemmler, Julian Rasch, and Dirk Ifenthaler. 2022. A Categorization of Workplace Learning Goals for Multi-Stakeholder Recommender Systems: A Systematic Review. *TechTrends* (2022), 1–14.

[20] Yu Jin, Andreas Loukas, and Joseph JaJa. 2020. Graph coarsening with preserved spectral properties. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 4452–4462.

[21] Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. 2020. Hard negative mixing for contrastive learning. *Advances in Neural Information Processing Systems* 33 (2020), 21798–21809.

[22] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *ICDM*. IEEE, 197–206.

[23] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems* 30 (2017).

[24] David Kempe, Jon Kleinberg, and Amit Kumar. 2000. Connectivity and inference problems for temporal networks. In *STOC*. 504–513.

[25] Joseph A Konstan, Ajith Muralidharan, Ankan Saha, Shilad Sen, Mengting Wan, and Longqi Yang. 2022. RecWork: Workshop on Recommender Systems for the Future of Work. In *RecSys*. 675–677.

[26] Gueorgi Kossinets, Jon Kleinberg, and Duncan Watts. 2008. The structure of information pathways in a social communication network. In *KDD*. 435–443.

[27] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (1978), 558–565.

[28] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. 2018. Graph summarization methods and applications: A survey. *Comput. Surveys* 51, 3 (2018), 1–34.

[29] Yifei Ma, Balakrishnan Narayanaswamy, Haibin Lin, and Hao Ding. 2020. Temporal-contextual recommendation in real-time. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 2291–2299.

[30] Friedemann Mattern. 1989. Virtual Time and Global States of Distributed Systems. In *Workshop on Parallel and Distributed Algorithms*.

[31] Adam Meyerson and Brian Tagiku. 2009. Minimizing average shortest path distances via shortcut edge addition. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 272–285.

[32] Tapan Mitra and Kemal Ozbek. 2021. Ranking by weighted sum. *Economic Theory* 72, 2 (2021), 511–532.

[33] Vincenzo Nicosia, John Tang, Cecilia Mascolo, Mirco Musolesi, Giovanni Russo, and Vito Latora. 2013. Graph metrics for temporal networks. In *Temporal Networks*. Springer, 15–40.

[34] Vincenzo Nicosia, John Tang, Mirco Musolesi, Giovanni Russo, Cecilia Mascolo, and Vito Latora. 2012. Components in time-varying graphs. *Chaos* 22, 2 (2012), 023101.

[35] Nikos Parotsidis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2016. Centrality-aware link recommendations. In *WSDM*. 503–512.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* 32 (2019).

[37] Zhen Qin, Le Yan, Honglei Zhuang, Yi Tay, Rama Kumar Pasumarthi, Xuanhui Wang, Mike Bendersky, and Marc Najork. 2021. Are neural rankers still outperformed by gradient boosted decision trees?. In *ICLR*.

[38] Anatol Rapoport. 1953. Spread of information through a population with socio-structural bias: I. Assumption of transitivity. *The Bulletin of Mathematical Biophysics* 15, 4 (1953), 523–533.

[39] Martin Riedmiller and Heinrich Braun. 1993. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *ICON*. IEEE, 586–591.

[40] Joshua David Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. 2020. Contrastive Learning with Hard Negative Samples. In *ICLR*.

[41] Daniel Romero and Jon Kleinberg. 2010. The directed closure process in hybrid social-information networks, with an analysis of link formation on twitter. In *ICWSM*, Vol. 4. 138–145.

[42] Javier Sanz-Cruzado and Pablo Castells. 2018. Enhancing structural diversity in social networks by recommending weak ties. In *RecSys*. 233–241.

[43] Qi Song, Yi Wang, Yang Chen, Jose Benitez, and Jiang Hu. 2019. Impact of the usage of social media in the workplace on team and employee performance. *Information & Management* 56, 8 (2019), 103160.

[44] Jean-François Stich, Monideepa Tarafdar, and Cary L Cooper. 2018. Electronic communication in the workplace: boon or bane? *Journal of Organizational Effectiveness: People and Performance* (2018).

[45] John Tang, Mirco Musolesi, Cecilia Mascolo, and Vito Latora. 2009. Temporal distance metrics for social network analysis. In *WOSN*. 31–36.

[46] Haoye Tian, Haini Cai, Junhao Wen, Shun Li, and Yingqiao Li. 2019. A music recommendation system based on logistic regression and eXtreme gradient boosting. In *2019 International Joint Conference on Neural Networks*. IEEE, 1–6.

[47] Antonela Tommasel, Juan Manuel Rodriguez, and Daniela Godoy. 2021. I Want to Break Free! Recommending Friends from Outside the Echo Chamber. In *RecSys*. 23–33.

[48] Sotiris Tsioutsiouliklis, Evaggelia Pitoura, Konstantinos Semertzidis, and Panayiotis Tsaparas. 2022. Link Recommendations for PageRank Fairness. In *WebConf*. 3541–3551.

[49] Thea Turner, Pernilla Qvarfordt, Jacob T Biehl, Gene Golovchinsky, and Maribeth Back. 2010. Exploring the workplace communication ecology. In *CHI*. 841–850.

[50] Ellen M Voorhees et al. 1999. The TREC-8 question answering track report.. In *TREC*, Vol. 99. 77–82.

[51] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. 2014. Path problems in temporal graphs. *Proceedings of the VLDB Endowment* 7, 9 (2014), 721–732.

[52] Lianghao Xia, Chao Huang, Yong Xu, and Jian Pei. 2022. Multi-Behavior Sequential Recommendation with Temporal Graph Transformer. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[53] Longqi Yang, David Holtz, Sonia Jaffe, Siddharth Suri, Shilpi Sinha, Jeffrey Weston, Connor Joyce, Neha Shah, Kevin Sherman, Brent Hecht, et al. 2022. The effects of remote work on collaboration among information workers. *Nature Human Behaviour* 6, 1 (2022), 43–54.

[54] Zhi Yu, Can Wang, Jiajun Bu, Xin Wang, Yue Wu, and Chun Chen. 2015. Friend recommendation with content spread enhancement in social networks. *Information Sciences* 309 (2015), 102–118.

[55] Eva Zangerle and Christine Bauer. 2022. Evaluating Recommender Systems: Survey and Framework. *Comput. Surveys* (2022).

[56] Yong Zheng and David Xuejun Wang. 2022. A survey of recommender systems with multi-objective optimization. *Neurocomputing* 474 (2022), 141–153.

[57] Xiaotian Zhou and Zhongzhi Zhang. 2021. Maximizing Influence of Leaders in Social Networks. In *KDD*. 2400–2408.

[58] Liwang Zhu, Qi Bao, and Zhongzhi Zhang. 2021. Minimizing Polarization and Disagreement in Social Networks via Link Recommendation. *Advances in Neural Information Processing Systems* 34 (2021), 2072–2084.

A WEIGHT CALIBRATION FOR TOTAL INFORMATION

We assign a single weight w_i to each edge type i (in our data, $i \in$ {post, reply, chat, email, file}) and fix λ so that 20% of information decays per day; with t in seconds, $\lambda = \exp(\log 0.8/(60 * 60 * 24)) \approx 0.999997$. We make the following assumptions: (1) the amount of information transmitted on each platform is the same, (2) information quantities are stable in the long run, (3) the stable amount of information a node has about its closest contact on a single platform is $\beta = 0.1$, and (4) if two nodes u, v both have mutually high information with a third node w, then u and v have high information about each other (the principle of *triadic closure* [15, 38, 41]).

Each day, we need the amount of information received by a node about their closest contact on each platform to result in stable information β after decay. To calibrate w_i to the edge frequency of the platform, we use two empirical measures: μ_i, the average number of edges received per day by nodes on platform i and ξ_i, the average number of edges received per day from each node's most frequent contact on platform i. A node u's closest contact v has information 1 about itself, and we assume all other nodes communicating with u have information β about v (i.e., we generously apply the triadic closure assumption). To simplify matters, we also assume all information arrives at the very start of the day, so that all decay for a day occurs after information has arrived. We then arrive at the following fixed point formula for the stable information β in terms of λ, μ_i, ξ_i, and the calibrated edge weight w_i:

$$\lambda^{60*60*24}(\underbrace{\beta}_{\text{prior inf.}} + w_i(\underbrace{\xi_i}_{\text{closest contact}} + \underbrace{\beta(\mu_i - \xi_i)}_{\text{other nodes}})) = \beta \qquad (4)$$

For each of the five edge types $i \in$ {post, reply, chat, email, file}, we then use empirical values of μ_i and ξ_i along with our assumed values of λ and β to derive a calibrated edge weight w_i. Edge weights used in our experiments are below.

	level 0	**level 1**	**org chart**
post	1.23×10^{-3}	9.96×10^{-3}	1.92×10^{-3}
reply	1.80×10^{-3}	1.34×10^{-2}	2.77×10^{-3}
chat	5.14×10^{-6}	8.47×10^{-5}	1.02×10^{-5}
email	1.60×10^{-4}	1.45×10^{-3}	2.20×10^{-4}
file	2.08×10^{-5}	4.80×10^{-4}	3.08×10^{-5}

B MAXIMUM INFORMATION

We define the *maximum information* v has about u at time t as the maximum age-discounted information transmitted along any path from u to v arriving no later than t:
$\text{maxInf}(u, v, t) = \max_{P \in \mathcal{P}_t(u,v)} \lambda^{t-d(P)} \prod_{e \in P} w_e$.

Theorem B.1. *We can compute all single-source maximum informations in time $O(m + nc \log n + nc \log c)$, where $n = |V|, m = |E|$, and $c = O(m)$ is the maximum degree of any node in G.*

See the online supplement for a proof (https://github.com/tomlinsonk/network-rec-supplement).

C TRAINING AND DATA DETAILS

We define the *recency* of an event that occurred t seconds ago (e.g., post creation) to be $\log^{-1}(2 + t)$ (or 0 if the event has not occurred). We construct the following features for the post ranking task for post p written by author v in channel c being recommended to user u: p creation recency, p action count, p action recency, u-p action count, u-p action recency, c action count, c action recency, c post count, u-c action count, u-c action recency, action count on posts by v, v post count. Here, "action" refers to replies and reacts. We log-transform all count features, adding 1 to avoid taking $\log(0)$. We implemented the (multinomial) *logit* recommender in PyTorch [36]. Given the set of up to 100 candidate posts to rank, the model predicts which one the user actually interacted with. We train the logit model with a negative log-likelihood (i.e., cross-entropy) loss. Posts are then ranked by their logit score. We trained the logit model with Rprop [39] with learning rate 0.05 for 100 epochs or until the squared gradient magnitude dropped below 10^{-8}. We use Rprop since it converges very fast for batch training without requiring learning rate tuning [10]. For the *lightgbm* recommender, we used the LightGBM binary classifier [23] with 5-round early stopping and fully default parameters. Here, the task is to predict whether a post is a positive (true interaction) or negative (other candidate post). Posts are then ranked by their LightGBM prediction score. We used the first 25% of edges for burn-in, the next 30% of edges for model training and validation, and the final 45% for testing.

D ADDITIONAL PLOTS

See the following page for versions of Figure 3 with the LightGBM recommender (Figure 6), with latency instead of total information (Figure 7), and with action probability $\rho = 1$ (Figure 8). Note that latency is a much coarser and noisier measure than total information.

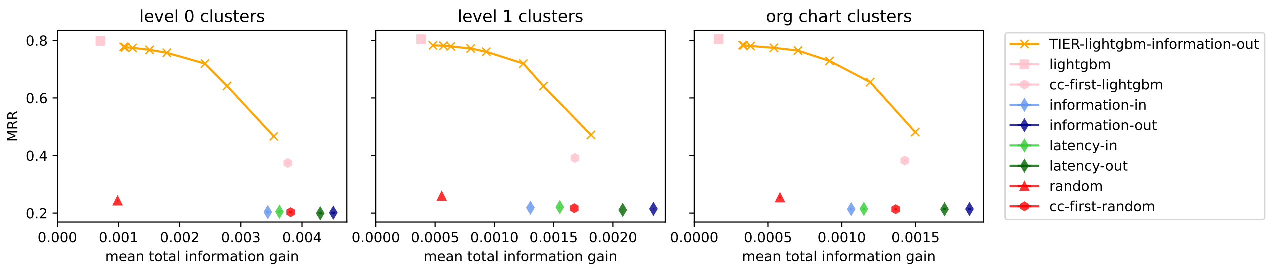

Figure 6: Relevance vs total information tradeoff ($\rho = 0.01$) for lightgbm recommenders.

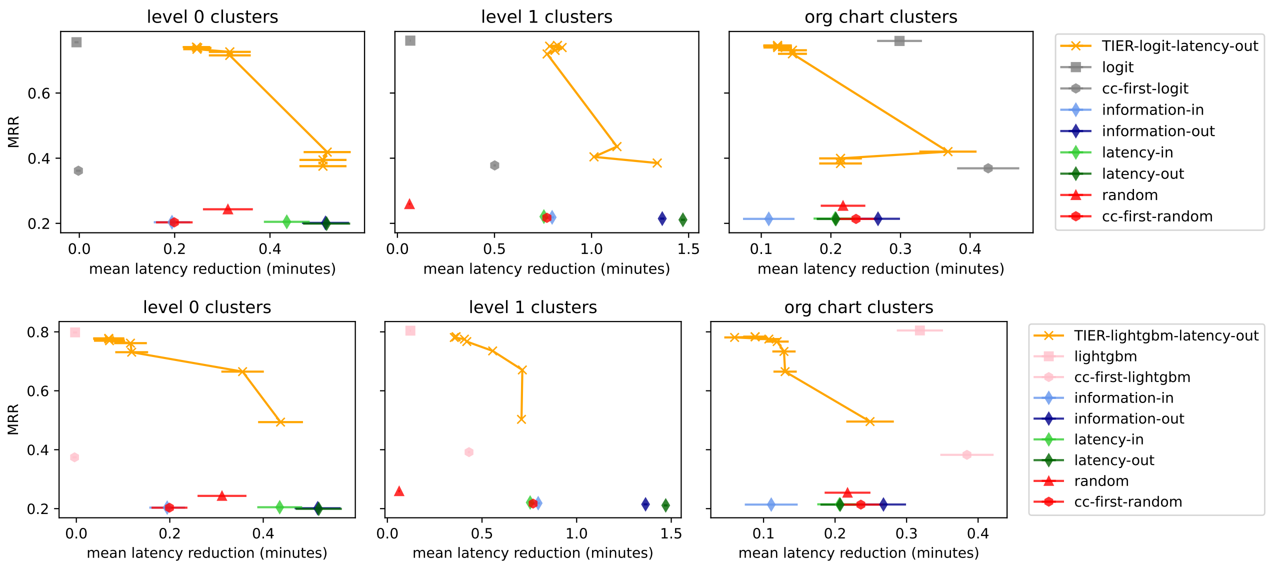

Figure 7: Relevance vs latency tradeoff ($\rho = 0.01$) for logit (top row) and lightgbm (bottom row) recommenders.

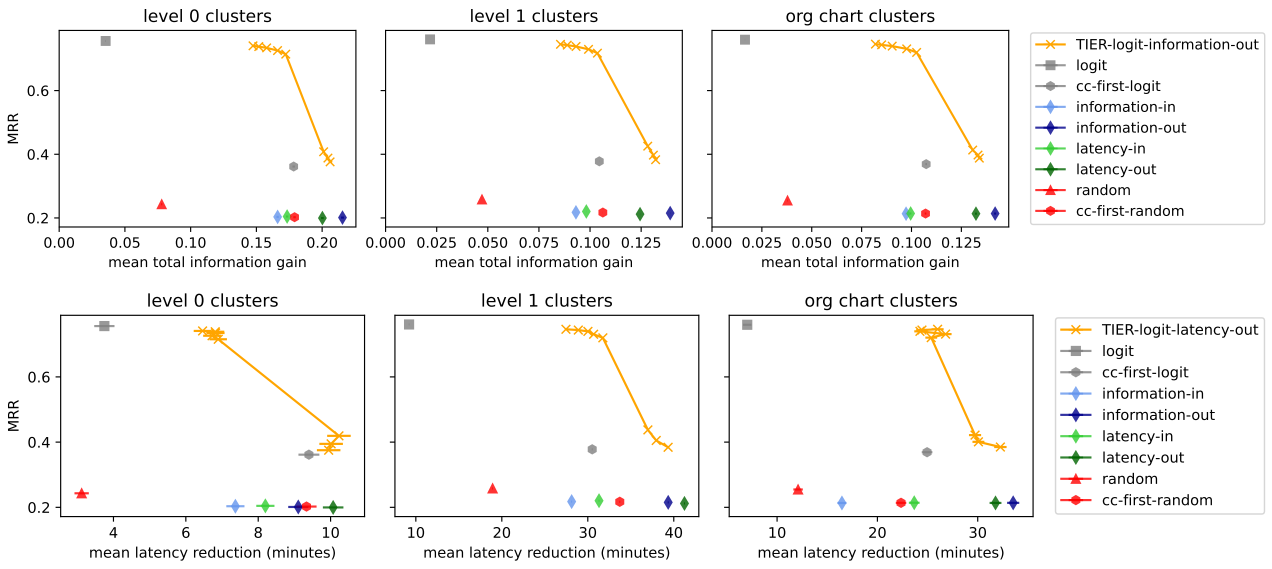

Figure 8: Relevance vs total information (top row) and latency (bottom row) tradeoff ($\rho = 1$) for logit recommenders.