Geoinformatica manuscript No.
(will be inserted by the editor)

MobiFeed: A Location-Aware News Feed Framework for Moving Users

Wenjian Xu · Chi-Yin Chow · Man Lung Yiu · Qing Li · Chung Keung Poon

Received: date / Accepted: date

Abstract A location-aware news feed system enables mobile users to share geo-tagged user-generated messages, e.g., a user can receive nearby messages that are the most relevant to her. In this paper, we present MobiFeed that is a framework designed for scheduling news feeds for mobile users. MobiFeed consists of three key functions, *location prediction*, *relevance measure*, and *news feed scheduler*. The *location prediction* function is designed to estimate a mobile user's locations based on a path prediction algorithm. The *relevance measure* function is implemented by combining the vector space model with non-spatial and spatial factors to determine the relevance of a message to a user. The *news feed scheduler* works with the other two functions to generate news feeds for a mobile user at her current and predicted locations with the best overall quality. We propose a heuristic algorithm as well as an optimal algorithm for the location-aware *news feed scheduler*. The performance of MobiFeed is evaluated through extensive experiments using a real road map and a real social network data set. The scalability of MobiFeed is also investigated using a synthetic data set. Experimental results show that MobiFeed obtains a relevance score two times higher than the state-of-the-art approach, and it can scale up to a large number of geo-tagged messages.

Wenjian Xu
Department of Computer Science, City University of Hong Kong, Hong Kong
E-mail: wenjianxu2-c@my.cityu.edu.hk

Chi-Yin Chow
Department of Computer Science, City University of Hong Kong, Hong Kong
E-mail: chiychow@cityu.edu.hk

Man Lung Yiu
Department of Computing, Hong Kong Polytechnic University, Hong Kong
E-mail: csmlyiu@comp.polyu.edu.hk

Qing Li
Department of Computer Science, City University of Hong Kong, Hong Kong
E-mail: itqli@cityu.edu.hk

Chung Keung Poon
School of Computing and Information Sciences, Caritas Institute of Higher Education, Hong Kong
E-mail: ckpoon@cihe.edu.hk

Keywords Location-aware news feed · Location-aware social networks · User mobility · Online scheduling · Optimization

1 Introduction

Social network systems, e.g., Facebook and Twitter, have become one of the major Web-based applications. They provide platforms for users to share user-generated multimedia messages and interact with their friends. With the advance in wireless communication and GPS-enabled mobile devices, social network systems have recently become location-aware, e.g., Facebook [23] and FourSquare [25]. Such applications provide new platforms for mobile users to share their locations and geo-tagged user-generated messages with their friends at anytime, anywhere.

A news feed is a common functionality of existing location-aware social network systems. It enables mobile users to post geo-tagged messages and receive nearby user-generated messages, e.g., "*Alice can receive 4 messages that are the most relevant to her among the messages within 1 km from her location every 10 seconds*". Fig. 1 depicts an application scenario. A mobile user, Alice, can generate a message and tag a point (e.g., m_1), a spatial extent (e.g., m_{14} is associated with a circular spatial area), or a venue (e.g., m_6 and m_7 are spatially associated with restaurant R_1) as its geo-location. Alice can also issue a location-aware news feed query to retrieve the k most relevant messages within her specified range distance D from her location.

The state-of-the-art research prototype of a location-aware news feed system is GeoFeed [9]. GeoFeed focuses on optimizing *static* queries over a set of registered locations (e.g., home and office) by deciding whether to pre-compute the k most *recent* messages within a user-specified distance of a registered location for an offline user. However, moving users have to keep issuing on-demand pull-based queries to GeoFeed to retrieve new news feeds. In general, GeoFeed has three drawbacks: (1) The moving user has to decide when she should update her location and send a new query to the server. (2) GeoFeed does not consider user preferences (i.e., the relevance of a message to a user) during the process of generating news feeds. (3) Even if GeoFeed takes user preferences into account by selecting k most *relevant*

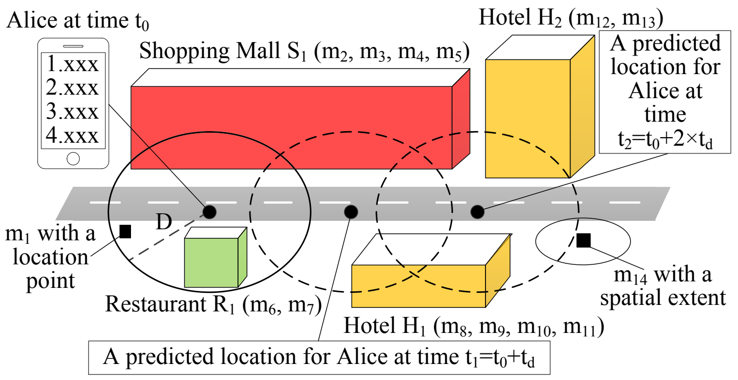

Alice at time t_0
1.xxx
2.xxx
3.xxx
4.xxx
Shopping Mall S_1 (m_2, m_3, m_4, m_5)
Hotel H_2 (m_{12}, m_{13})
A predicted location for Alice at time $t_2=t_0+2\times t_d$
D
m_1 with a location point
Restaurant R_1 (m_6, m_7)
Hotel H_1 (m_8, m_9, m_{10}, m_{11})
m_{14} with a spatial extent
A predicted location for Alice at time $t_1=t_0+t_d$

Fig. 1: Location-aware news feed scheduling.

messages as a news feed for each query region, the overall quality of news feeds is not optimized. This is because the news feeds are only computed based on a user's location at the query time (i.e., it does not consider the user's future locations). For example, in Fig. 1, there are 14 messages (i.e., m_1 to m_{14}) with their geo-location intersecting Alice's query regions at time t_0, t_1, and/or t_2. Assume m_i is more relevant to Alice than m_j if $i < j$. GeoFeed returns (m_1, m_2, m_3, m_4) at t_0, (m_5, m_8, m_9, m_{10}) at t_1, and $(m_{11}, m_{12}, m_{13}, m_{14})$ at t_2. However, if we consider all three query regions at the same time, a better solution returns (m_1, m_2, m_6, m_7) at t_0, (m_3, m_4, m_5, m_8) at t_1, and $(m_9, m_{10}, m_{11}, m_{12})$ at t_2 because m_6 and m_7 are more relevant to Alice than m_{13} and m_{14}.

In this paper, we present MobiFeed that is a location-aware news feed framework designed for social network systems to schedule news feeds for mobile users. MobiFeed consists of three key functions: *location prediction*, *relevance measure*, and *news feed scheduler*. As shown in Fig. 1, given a user u's location *u.location* at current time t_0, u's required minimum message display time t_d, u's specified range distance D, u's requested number of messages per news feed, and a look-ahead step n, the *location prediction* function estimates n future locations for u at times $t_1 = t_0 + t_d$, $t_2 = t_0 + 2 \times t_d$, $\dots$, and $t_n = t_0 + n \times t_d$, the *relevance measure* function calculates the relevance score of each candidate message with a geo-location intersecting any u's query region (i.e., a circular area centered at *u.location* or a predicted location with a radius D), and the *news feed scheduler* generates news feeds from the candidate messages for u's query regions at t_0, t_1, ..., and t_n with the highest overall relevance score. The computed $n + 1$ news feeds are sent to u. u's mobile device immediately displays the first news feed, i.e., the one with respect to the query region at t_0, and then displays each of the remaining news feeds one by one for every t_d. In contrast to GeoFeed, MobiFeed is equipped with a location-aware *news feed scheduler*, which works with the *location prediction* and *relevance measure* functions to provide high-quality news feeds for moving users.

Designing a scalable and effective *news feed scheduler* has several key challenges. (1) A message has a lifetime with respect to a user's movement. A message can be a candidate message for several *consecutive* or *non-consecutive* news feeds. The minimum display time periods of these news feeds constitute the message's lifetime (as shown in the timeline chart in Fig. 3). The scheduler should select at most k candidate messages for a news feed within their lifetime intervals such that the overall quality of a user's news feeds is maximized. (2) The relevance of a message to a user is highly dynamic. Since we consider the distance between a message and a user as one of the factors in the relevance measure, the relevance of a message could vary for a user at different locations. (3) A user prefers to have the most relevant message at the top of a result list. The relevance of a message displayed on a screen should be weighted by its position. For example, the highest weight is given to the message displayed at the top on the screen. (4) The online scheduler should be efficient such that it could scale up to a large number of messages.

The contributions of this paper are summarized as follows:

- We design the *location prediction* function based on the path prediction algorithm [30], and combine the vector space model [42] with spatial and non-spatial factors (e.g., message contents and categories) to define the *relevance measure* function.

- We incorporate location prediction to the process of location-aware news feed generation, thus formulating a novel n-look-ahead news feed scheduling framework to improve the overall quality of *multiple* news feeds for moving users.
- We propose a *heuristic* n-look-ahead news feed scheduler for the sake of efficiency. Moreover, we present an *optimal* scheduler by finding the maximum weight matching in a weighted bipartite graph; we also provide correctness proof and complexity analysis for our optimal scheduler.
- We evaluate the performance of MobiFeed through extensive experiments based on a real location-aware social network data set and a real road map. We also study the scalability of MobiFeed using a synthetic data set. Experiment results show that MobiFeed usually obtains a relevance score two times higher than GeoFeed, and it can scale up to a large number of geo-tagged messages.

The rest of this paper is organized as follows. Section 2 highlights related work. The MobiFeed framework is illustrated in Section 3. The *location prediction* and *relevance measure* functions are described in Section 4. Sections 5 and 6 present our heuristic and optimal *news feed schedulers*, respectively. We show the complexity of two schedulers in Section 7. Section 8 analyzes experimental results. Finally, Section 9 concludes this paper.

2 Related Work

News feed systems. Most existing news feed systems only provide publish/subscribe services that simply forward a message to subscribed users or friends, e.g., [13, 14, 51, 65]. However, such systems are not applicable to location-aware news feeds because they ignore the spatial relevance of messages, and they simply push all new messages to their subscribers without taking the subscriber's capacity or preferences into account, and they only consider stationary users.

Location-aware social networks. Table 1 compares the features of MobiFeed with existing commercial products and research prototypes. There are two major categories for existing commercial products. Facebook [23], Renren [49], and Sina Weibo [52] belong to the first category, where a message is tagged with its issuer's location, but users may receive the same geo-tagged news feeds regardless of their location. Loopt [40], Google Buzz Mobile [27], Foursquare [25], and Twinkle [54] belong to the second category, where each message has location tags and a user

Table 1: The key features of MobiFeed.

Location-Aware Social Network	Location Tags	Range Queries	Spatial Messages	Location Prediction & Relevance Measure	News Feed Scheduling
Facebook [23]	√				
Renren [49]	√				
Sina Weibo [52]	√				
Loopt [40]	√	√			
Google Buzz [27]	√	√			
Foursquare [25]	√	√			
Twinkle [54]	√	√			
GeoFeed [9]	√	√	√		
MobiFeed	√	√	√	√	√

can issue a range query to view messages within a certain distance from her location. GeoFeed [9] is the state-of-the-art research prototype that further enables a message to be associated with a spatial extent to control where users can receive it. However, GeoFeed supports mobile users in a very limited extent because it only optimizes the system performance for users with a set of registered locations (e.g., home and office). We can distinguish MobiFeed from these systems as it employs *location prediction* and *relevance measure* functions as well as heuristic and optimal *news feed schedulers* to schedule location-aware news feeds for moving users.

Related research work in location-aware social network systems have focused on three major directions. (a) *Message sharing.* Some systems enable mobile users to broadcast or receive public geo-tagged messages, but they do not consider any non-spatial aspects (e.g., user preferences) or schedule messages for mobile users based on their movements [2,8,11]. (b) *Privacy-preserving location sharing.* Users can share their locations with friends without revealing any location information to the social network system or other unauthorized users [26, 35, 50]. (c) *Location-aware recommendation.* In existing location-aware recommender systems, mobile users receive suggestions for new places/activities [8, 37, 38, 58, 60, 62, 63], friends [5, 64] or articles [9, 53] based on user preferences, spatial-temporal proximity, and social influence, etc. Although their recommendation techniques can be used to suggest news feeds for users, the overall quality of news feeds is not optimized if we just apply these techniques to our problem. This is because the news feeds are only computed based on a user's location at the query time (i.e., it does not consider the user's future locations). To this end, we incorporate the *location prediction* function to the process of location-aware news feed recommendation, thus formulating a novel n-look-ahead framework to improve the overall quality of generated news feeds for moving users. Experiment results in Section 8 show that our framework generates news feeds with higher quality than a state-of-the-art location-aware recommender system (i.e., [9]).

Spatial-textual query processing. As the geo-tagged messages in our framework are *spatially* and *textually* relevant to the querying users (see Section 4.2), we consider spatial-textual (or spatial keyword) query processing [12, 19, 21, 61] as our related work. Spatial-textual queries utilize some efficient index structures (e.g., IR-tree and its variants [19, 39, 56]) to retrieve a ranked list of objects according to their joint spatial and textual relevance to the query. However, their objective is to efficiently compute exact *top-k* results for *individual* query points; in contrast, our news feed scheduling framework aims at maximizing the overall quality of *multiple* news feeds for *moving* users. Similarly, the safe-zone methods proposed for *continuous* spatial-textual queries [57] are also not applicable to our scenario because of their limitation to top-k semantics. Furthermore, we argue that the index structures of above-mentioned spatial-textual query processing techniques *cannot* be applied to our n-look-ahead schedulers. The reason is that, given a news feed query, our schedulers require $n + 1$ *complete* sets of candidate messages along with their relevance scores, rendering the pruning effect of those indexing techniques useless.

Online scheduling algorithms. Our problem of scheduling location-based news feeds for moving users can be formulated as an online scheduling problem as follows [20]. Each position of a news feed result list is modeled as a processor, and each message corresponds to a job weighted according to its relevance to the user and has arrival time and deadline determined by its location information and the moving user's trajectory. Further, as shown in our running example (Figure 3), it is possible that a job is unavailable for a certain period and a job has different weights to the same user at different locations. Since all messages are to be displayed for the same user-specified minimum display time, the corresponding jobs have the same processing time. By proper scaling, the jobs have unit processing times and integral arrival times and deadlines. When a job is processed, we gain a profit that is equal to the product of the weight of the job and the weight of the assigned processor. Our objective is to maximize the total gain of the processed jobs. Note that our problem has an on-line flavor because we do not know of a job (message) until it arrives. Once we receive a job, we can predict its deadline and unavailable periods by predicting the moving user's trajectory. There are other similar but different problems studied in literature. For example, Chin et al. [16] presented an on-line algorithm for the special case when all the processors are identical and there is no job unavailable period. Chen et al. [15] gave approximation algorithms for maximizing the number of completed jobs with multiple feasible intervals but they considered unweighted jobs on a single processor. To our best knowledge, none of the existing work studied the problem we formulated here. We aim at developing efficient heuristic and optimal news feed scheduling algorithms. Moreover, we evaluate its effectiveness by extensive experiments using a real road map and a large-scale social network data set.

Bipartite matching. Our problem of scheduling location-based news feeds for moving users is related to bipartite matching problems [33, 36, 41]. We first review offline bipartite matching algorithms. Then we discuss their online counterparts. Finally, we explain the reason why we apply *offline* algorithms to solve our news feed scheduling problem.

Given a bipartite graph $G(V, E, W)$, where V is a vertex set, E is an edge set, and W is the maximum weight of the edges in E, a *maximum weight matching* is defined as a matching where the sum of the values of the edges in the matching have a maximal value [55]. Finding such a matching *offline* is also known as the *assignment problem*. The remarkable Hungarian algorithm [36] solves the offline bipartite matching problem, which uses a modified shortest path search in the augmenting path algorithm [55]. Its time complexity is $O(|V|^4)$. Kao et al. [31, 32] present a decomposition theorem and use it to design an $O(\sqrt{|V|} \cdot |E| \cdot W \cdot log_{|V|}(\frac{|V|^2}{|E|}))$-time algorithm for computing a maximum weight matching. In practice, the Hungarian algorithm can be improved to $O(|V|^3)$ [55], which outperforms the algorithms in [31, 32] when $|V|$ is much smaller than $|E|$ and W; thus, we use the improved Hungarian algorithm for our bipartite matching problem.

Next, we discuss the *online* bipartite matching algorithm, which was first studied by Karp et al. [33]. Given a bipartite graph $G(U, V, E)$ in this problem, one side U is known to us in advance and the vertices of the other V arrive online, one vertex at a time. Whenever a vertex $v \in V$ arrives, its adjacent edges connecting to U are revealed. The online algorithm has to decide which of these edges should be included in the matching. A match once made cannot be revoked. The objective is to maximize the size of the matching obtained after all vertices in V arrive. In [43], Mehta provides a classification of main generalizations of online bipartite matching problems and their applications:

- **Online vertex-weighted bipartite matching:** In this problem, each vertex $u \in U$ has a non-negative weight w_u, and the goal is to maximize the sum of weight of vertices in U which are matched. This problem was introduced by Aggarwal et al. [4], and its typical application is *online budgeted allocations* [44] in the case when an agent makes the same bid for any desired item.
- **Online edge-weighted bipartite matching:** In this problem, each vertex $u \in U$ has a capacity c_u, which is an upper bound on how many vertices $v \in V$ can be matched to u. Each edge $(u, v) \in E$ has a weight w_{uv}. The goal is to maximize the total weight of edges matched. Its application includes assigning ad impressions to advertisers online [24] and matching applicants to possible positions in a company (i.e., *secretary matching problem*) [34].
- **AdWords:** This problem formulated by Mehta et al. [45] is closely related to google's online advertising product AdWords [3]. Each vertex $u \in U$ has a budget B_u, and each edge $(u, v) \in E$ has bids bid_{uv}. By matching an arriving vertex v to a neighbour u, u consumes bid_{uv} amount of its budget. After u uses up its entire budget, it becomes unavailable. The objective is to maximize the total budget consumed.

Although we design an online location-based news feed scheduler for MobiFeed, its bipartite matching uses an offline algorithm (see Section 6.1). This is because online bipartite matching algorithm assumes that vertices come in an unpredicted way, but in our bipartite matching problem, all vertices (a querying user's reported location and n predicted locations) are already known before the bipartite matching algorithm starts. In MobiFeed, we employ the state-of-the-art implementation of the Hungarian algorithm (i.e., [55]) to solve our news feed scheduling problem.

3 System Overview

Fig. 2 depicts an overview of the MobiFeed framework. MobiFeed stores geo-tagged user-generated messages in a database. It is equipped with a *news feed scheduler* which interacts with the *location prediction* and *relevance measure* functions to select a collection of messages from the database as a news feed for a moving user at a particular location.

Geo-tagged messages. A geo-tagged user-generated message is defined as a tuple (*MessageID*, *SenderID*, *Content*, *Timestamp*, *Category*, *Spatial*), where *MessageID* is its identifier, *SenderID* is its sender's identifier, *Content* is its content, *Timestamp* is

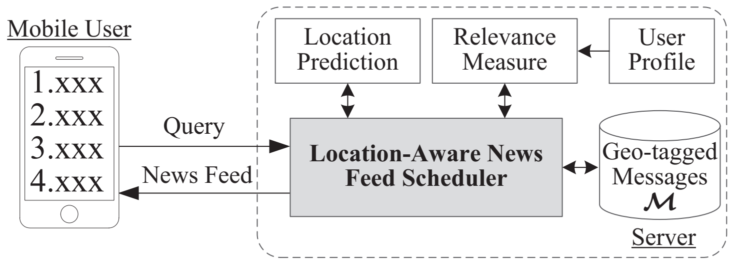


Fig. 2: MobiFeed framework.

its submission time, and *Spatial* specifies its spatial extent. As depicted in Fig. 1, the spatial extent of a message can be a point location (e.g., m_1), a user-specified spatial region (e.g., m_{14}), or the spatial region of a venue (e.g., the spatial extent of m_2 is the shopping mall S_1). For the *Category* attribute, we group messages into categories based on their geo-tagged locations or keywords. For example, in FourSquare [25], each message is categorized by its associated venue, e.g., restaurant and museum.

System users and news feed queries. In MobiFeed, a mobile user u is able to (a) post a new message with a spatial extent, and (b) receive at most $u.k$ messages within u's specified range distance $u.D$ (i.e., the query region of a news feed) at a particular time as a news feed through a GPS-enabled mobile device. Specifically, u issues news feed queries by specifying the news feed size ($u.k$), the range distance ($u.D$), and the minimum message display time ($u.t_d$). Then, MobiFeed computes news feeds for u by selecting messages based on their relevance to u and u's movement. Each selected message must be displayed on u's mobile device without any interruption for at least u's specified minimum display time $u.t_d$. Assume the look-ahead step is n (system parameter), u reports its location to the server at every time period $(n+1) \times u.t_d$. After receiving u's location update, $n+1$ news feeds are computed for u. u's mobile device immediately displays the first news feed, and then displays each of the remaining news feeds one by one for every $u.t_d$. It is important to notice that location-based services usually support the user-specified range distance $u.D$ because users would be more interested in nearby messages or events and $u.D$ can be used to prune the entire message set into a much smaller candidate message set for query processing (i.e., significantly improve the query processing performance).

Quality measure. Given a user u_i and a message m_j, the relevance measure function returns a relevance score $relevanceScore(u_i, m_j)$. This function considers u_i's preferences with regard to categories and keywords maintained in the *user profile* (Fig. 2), as well as the geographical proximity between u_i and m_j to define their relevance (details will be illustrated in Section 4.2). In information retrieval, query-relevance ranking algorithms usually display a document that is more relevant to a user's query at a higher position in a result list [7]. To this end, MobiFeed supports different weights for different positions in a news feed result list, i.e., a higher weight is given to a message displayed at a higher position because it would be easier to draw a user's attention. In this paper, we use a simple weighting scheme. Given a result list with k positions, the weight of the first position is k, the weight of the second position is $k-1$, and so on. In general, the weight of a message m_j at the j-th position $(1 \leq j \leq k)$ is $displayWeight(j,k) = k - (j-1)$. Thus, the relevance score of a news feed f_i with k messages $m_1, m_2, \ldots, m_k$ displayed at the j-th position in a result list for a user u_i is calculated as:

$$relevanceScore(f_i) = \sum_{j=1}^{k} relevanceScore(u_i, m_j) \times displayWeight(j,k) \qquad (1)$$

Problem definition. Our scheduling problem can be formulated as follows: Given a user u's location-aware news feed query at the current time t_0 and a look-ahead step n, MobiFeed predicts u's locations at each of the next n minimum display times (i.e., $t_0 + 1 \times u.t_d$, $t_0 + 2 \times u.t_d$, …, $t_0 + n \times u.t_d$), and schedules at most k

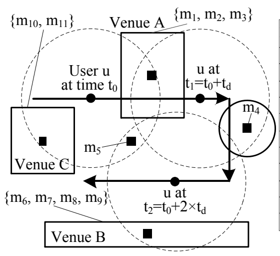


Table I. Non-spatial information

Message	Categories	Message content relevance to u
m_1	Restaurant	0.6
m_2	Restaurant	0.6
m_3	Restaurant	0.1
m_4	Stadium	0.2
m_5	Stadium	0.9
m_6	Shopping	0.2
m_7	Shopping	0.4
m_8	Shopping	0.6
m_9	Shopping	0.6
m_{10}	Museum	0.35
m_{11}	Museum	0.25

Table II. Spatial information

Time	Message	Normalized distance to u
t_0	Venue A	0.55
	Venue C	0.75
	m_5	0.15
t_1	Venue A	0.75
	m_4	0.6
t_2	Venue A	0.1
	Venue B	0.4
	m_4	0.1
	m_5	0.15

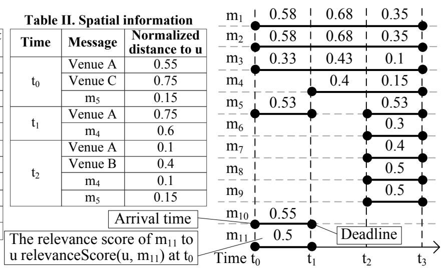


Fig. 3: Location-aware news feed scheduling.

messages (which have not been sent to u before) for the news feed at each location (i.e., one reported and n predicted locations), such that the total relevance score of the generated news feeds is maximized. In this paper, we design a heuristic news feed scheduler for efficiency and an optimal one for the best quality.

4 Location Prediction and Message Relevance Measure

In this section, we present the details of the *location prediction* and *relevance measure* functions in MobiFeed.

4.1 Location Prediction

The *location prediction* function can employ any existing location prediction algorithm if it can predict a user's location at a specified future time in a road network. We here describe how to incorporate the path prediction algorithm [30] into MobiFeed. Given a user u's current location, u's historical trajectories, the road map, and a future time t, the path prediction algorithm estimates u's location at t. Fig. 4 depicts a graph model $G = (V, E)$ of a road network, where E is a set of road segments and V is a set of intersections of road segments that are represented by circles and lines, respectively. The algorithm performs two steps to predict u's direction and speed.

Step 1. Direction prediction. When a user u is moving on an edge e_i, this step predicts which adjacent edge e_j of e_i u will go based on u's historical trajectory set $\mathcal{T}_u$. This step predicts a user's direction through one of the following three cases [30]: (1) Given two edges e_i and e_j incident to a vertex v, the transition probability of u turning from e_i to e_j is defined as $P(e_i, v, e_j) = \frac{\tau(\mathcal{T}_u, e_i \rightarrow e_j)}{\sum_{e_k} \tau(\mathcal{T}_u, e_i \rightarrow e_k)}$, where $\tau(\mathcal{T}_u, e_i \rightarrow e_j)$ is the number of trajectories in $\mathcal{T}_u$ that turn from e_i to e_j and e_k is an adjacent edge of e_i incident to v. For each adjacent edge e_j of e_i incident to v, this step calculates $P(e_i, v, e_j)$ and predicts that u will turn to e_j with the largest probability. For example, Fig. 4 shows a user u's four historical trajectories T_1 to T_4 and u is moving towards v_2 on e_7. Since $P(e_7, v_2, e_4) = 2/3$, $P(e_7, v_2, e_5) = 0$, and $P(e_7, v_2, e_6) = 1/3$, we predict that u will move to e_4. (2) However, if $\tau(\mathcal{T}_u, e_i \rightarrow e_j)$ is empty, the notion of reverse mobility statistics $P(e_i, v, e_j) = \frac{\tau(\mathcal{T}_u, e_j \rightarrow e_i)}{\sum_{e_k} \tau(\mathcal{T}_u, e_k \rightarrow e_i)}$ is used. (3) In case that both $\tau(\mathcal{T}_u, e_i \rightarrow e_j)$ and $\tau(\mathcal{T}_u, e_j \rightarrow e_i)$ are empty, we select the adjacent edge of e_i incident to v with the smallest deviation angle from u's current travel direction, which is derived from u's initial location at the query time and current location.

Step 2. Speed prediction. This step estimates u's travel speed $S(e)$ on an edge e by the average historical travel speeds of e from u's $\mathcal{T}_u$ [30]. If e does not exist in $\mathcal{T}_u$, we use a heuristic method that computes $S(e) = A(e) \times \alpha$, where $A(e)$ is the speed limit of e and α is a system parameter.

Let's consider a general case where u is moving on an edge e_i, and then u will enter an edge e_j at t' from a vertex v_s and stay at e_j at t. Let (x_s, y_s) denote the location of v_s and (x_e, y_e) denote the location of the other vertex of e_j. The predicted location of u at t is calculated as $(\frac{\lambda_1 \times x_e + \lambda_2 \times x_s}{\lambda_1 + \lambda_2}, \frac{\lambda_1 \times y_e + \lambda_2 \times y_s}{\lambda_1 + \lambda_2})$, where $\lambda_1 = (t - t') \times S(e_j)$ and $\lambda_2 = L(e_j) - \lambda_1$ (where $L(e_j)$ is the length of e_j).

4.2 Message Relevance Measure

MobiFeed only requires the *relevance measure* function to return a score to indicate the relevance of a message m_j to a user u_i, i.e., *relevanceScore*(u_i, m_j). We describe how to combine three relevance measure methods to implement the *relevance measure* function.

Message categories. The user usually has a preference for certain message categories. For instance, movie fans may prefer to receive movie showtimes from nearby cinemas, while shopaholics would be interested in discount information from nearby shopping malls [47]. To this end, we use the category information to measure the relevance of a message to a user. Specifically, for each user u_i in the *user profile* (see Fig. 2), we maintain a list of categories $CateList_i$ sorted by the number of u_i's submitted messages belonging to each category. For example, if u_i has issued three messages (i.e., m_1 and m_2 belong to the "restaurant" category, and m_3 belongs to the "stadium" category), then $CateList_i = [\ restaurant(2), stadium(1)\ ]$, where the integer value in the parentheses counts the number of messages associated with each category.

Message contents. The user may be more interested in messages that are similar to her submitted ones (e.g., a user's common keywords reflect her interests [48]). For example, a user issued a message "I like spicy food" would be happy to receive

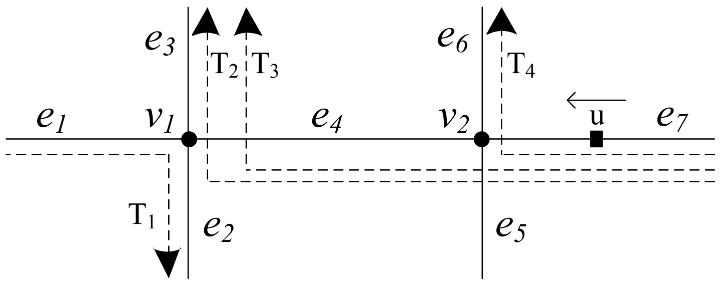


Fig. 4: Four historical trajectories of a user.

messages about Thai restaurants. Therefore, we use the vector space model [42] to measure the relevance of a message to a particular user in terms of content similarity. Specifically, let $\mathcal{T}$ denote the term set of the message set $\mathcal{M}$ (see Fig. 2), and each message m_j is represented by a vector of weights of all the terms in $\mathcal{T}$, i.e., $m_j.V = \langle w_{j1}, w_{j2}, \ldots, w_{j|\mathcal{T}|}\rangle$. In general, a term $T_k \in \mathcal{T}$ should be weighted higher for m_j if T_k occurs more frequently in m_j and occurs more rarely in other messages in $\mathcal{M}$. The weight can be computed by the $TF \times IDF$ scheme: $w_{jk} = tf_{jk} \cdot \log \frac{|\mathcal{M}|}{df_k}$, where tf_{jk} is the *term frequency* of T_k in m_j and df_k is the *document frequency* of T_k in $\mathcal{M}$. To incorporate the vector space model into MobiFeed, we maintain a preference vector in the *user profile* (see Fig. 2) for each user based on her submitted messages. Given a querying user u_i with a preference vector $u_i.V$ and a message m_j, we use the cosine similarity to compute $contentScore(u_i, m_j) = \frac{\sum_{k=1}^{|\mathcal{T}|} w_{ik} \cdot w_{jk}}{\sqrt{\sum_{k=1}^{|\mathcal{T}|} w_{ik}^2} \cdot \sqrt{\sum_{k=1}^{|\mathcal{T}|} w_{jk}^2}}$, where $w_{ik} \in u_i.V$ and $w_{jk} \in m_j.V$.

Distance. We argue that the geographical proximities between users and messages have a significant influence on the relevance measure. In our scenario, the relevance of a message m_j to a user u_i can be measured by their distance, i.e., $Dist(u_i, m_j)$. If m_j is associated with a spatial extent or a venue, $Dist(u_i, m_j)$ returns the *minimum* distance between u_i and the spatial extent or the venue of m_j. To accommodate the difference in the value ranges of $Dist(u_i, m_j)$ and other relevance measures, we normalize $Dist(u_i, m_j)$ to a range from zero to one (i.e., $NDist(u_i, m_j) = 1 - \frac{Dist(u_i, m_j)}{u.D}$), where $u.D$ is the query range distance of a news feed.

Relevance measure function. We employ two-level and linear combinations to integrate the aforementioned three methods into the *relevance measure* function. At the first level, we select top-δ categories $u_i.C$ in $CateList_i$ for a querying user u_i, where δ is a system parameter to specify the number of top categories. All messages within u_i's range distance $u_i.D$ and belong to categories in $u_i.C$ are considered as the set of candidate messages CM, i.e.,

$$CM = \{m | m \in \mathcal{M}, Dist(u_i, m) \leq u_i.D, m.Category \in u_i.C\} \tag{2}$$

At the second level, for each candidate message $m_j \in CM$, we measure its relevance to u_i using a linear combination of $contentScore(u_i, m_j)$ and $NDist(u_i, m_j)$ [17]:

$$relevanceScore(u_i, m_j) = contentScore(u_i, m_j) \times (1 - \beta) + NDist(u_i, m_j) \times \beta \tag{3}$$

where $0 \leq \beta \leq 1$ and β is a parameter that indicates the importance of the distance factor with respect to the message content score.

In our example (Fig. 3) where $\delta = 4$, $n = 2$ and $\beta = 0.5$, all candidate messages (i.e., m_1 to m_{11}) belong to the top-4 categories (i.e., restaurant, stadium, shopping, and museum). For the user u and each candidate message m_j, the values of $contentScore(u, m_j)$ and $NDist(u, m_j)$ are listed in Tables I and II, respectively. At the second level, we calculate the relevance score $relevanceScore(u, m_j)$ for the query regions at times t_0, t_1, and t_2. For instance, m_4 is the candidate message at t_1 and t_2, i.e., $relevanceScore(u, m_4) = 0.2 \times 0.5 + 0.6 \times 0.5 = 0.4$ at t_1 and $relevanceScore(u, m_4) = 0.2 \times 0.5 + 0.1 \times 0.5 = 0.15$ at t_2.

As depicted in Figure 2, the proposed location-aware news feed scheduler only relies on the relevance measure function to return the relevance of a message to a user as an input. The relevance measure function can be customized based on different application scenario. For instance, the cosine similarity measure can be replaced with other similarity measure models (e.g., a probabilistic model). Furthermore, the relevance measure function can consider additional factors. For example, in Section 8.3, we will consider a temporal factor as an additional aspect in Equation 3. The key idea behind the temporal factor is that the interestingness and importance of a message fade out as time goes on.

5 A Heuristic News Feed Scheduler

In this section, we present an n-look-ahead location-aware news feed scheduling algorithm for MobiFeed, where n is a system parameter to control the number of locations predicted for a mobile user. In general, if an underlying path prediction algorithm provides more accurate locations, we should specify a larger n. Thus, the value of n should be adjusted for different users and areas in a road network, for example, $P(e_i, v, e_j)$ should be larger than a certain threshold.

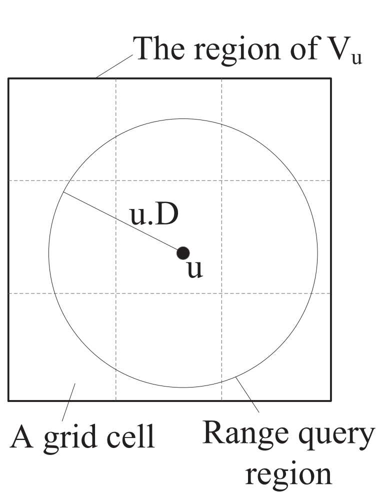
Data structure.. In MobiFeed, a spatial grid structure is used to index all geo-tagged messages, i.e., each grid cell stores messages with spatial extent intersecting its cell area. Given a user u's news feed query, a range query is issued to the grid index to retrieve the messages, which are not generated by u, associated with spatial extent intersecting the query region.

Algorithm. After a mobile user u issues a location-aware news feed query to MobiFeed, the *location prediction* function returns n predicted locations for u. A set of candidate messages is found for each of $n+1$ locations. The *relevance measure* function filters out those candidate messages that do not belong to any u's top-δ categories, and then determines the relevance of each remaining candidate message to u. The scheduler finally computes a news feed for each location such that the overall relevance score is maximized. The heuristic n-look-ahead scheduling algorithm has three main steps.

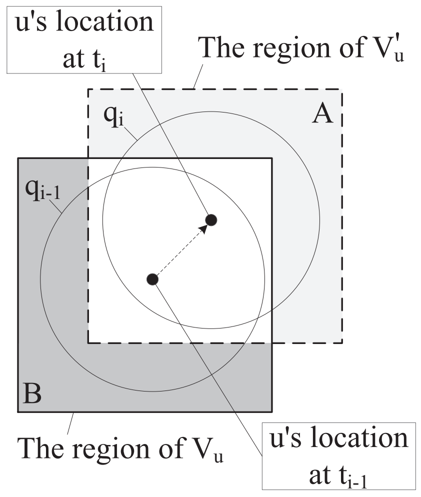
Step 1. Candidate message step. Given u's query at time t_0, the *location prediction* function predicts n locations for u at times t_1, t_2, $\ldots$, t_n, where $t_i = t_0 + u.t_d \times i$ and $u.t_d$ is u's user-specified message minimum display time. For each of $n+1$ locations, a range query with a circular region centered at the location with a radius of $u.D$ is issued to retrieve the messages intersecting the query region as a set of candidate messages CM_i $(0 \leq i \leq n)$. If there is an overlapping area between two consecutive query regions, we can employ a materialized view V_u for u to reduce the query processing time. The region of a materialized view of a range query is a minimum bounding rectangle of all the grid cells intersecting the range query region, as depicted in Fig. 5a.

Given a range query q_i at time t_i, where $0 \leq i \leq n$, this step first finds a set of candidate messages for q_i (i.e., CM_i) as follows:

- When $i = 0$, we find the region of V_u for q_i's query region (Fig. 5a). The messages intersecting the region of V_u are materialized in V_u and then V_u is used to compute CM_i for q_i.
- When $i > 0$, a materialized view V_u has been computed for q_{i-1} at time t_{i-1}, where $0 < i \leq n$. We first compute the region of a new materialized view V'_u for q_i. If the query regions of q_{i-1} and q_i overlap (Fig. 5b), V'_u can be efficiently computed by (i) removing a message m from V_u, if m's spatial extent



(a) Range query



(b) Incremental query processing

Fig. 5: Materialized view.

is completely within the non-overlapping area of q_{i-1}'s V_u (i.e., region B), and (ii) adding a message m to V_u, if m's spatial extent intersects the non-overlapping area of V_u''s region (i.e., region A) but not V_u's region. V_u' is then used to compute CM_i.

The experiment results show that using materialized views can significantly improve the query processing time, as depicted in Fig 10b in Section 8.

After retrieving CM_i, the *relevance measure* function filters out all messages that do not belong to any u's top δ categories. For each remaining candidate message m, a relevance score for u, i.e., *relevanceScore*(u, m), is calculated to indicate the relevance of m to u. Finally, the messages in each CM_i are sorted by their relevance score in non-increasing order. To break ties, precedence will be given to a message with a more recent post time. The pseudo code of this step is shown in Lines 4 to 17 in Algorithm 1.

Example. Fig. 3 depicts an example of location-aware news feed scheduling, where a user u sends a query with her location to the server at the current time t_0. The *location prediction* function predicts u's location at each of the next two (i.e., $n = 2$) minimum display times t_d, i.e., $t_1 = t_0 + t_d$ and $t_2 = t_0 + 2 \times t_d$. There are totally 11 candidate messages for the three news feeds at the times t_0, t_1, and t_2. m_4 and m_5 are tagged with a spatial region and a point location, respectively. $\{m_1, m_2, m_3\}$, $\{m_6, m_7, m_8, m_9\}$, and $\{m_{10}, m_{11}\}$ are associated with venues A, B, and C (represented by rectangles), respectively. The lifetime of each message with its relevance score for u at t_0, t_1, and t_2 is shown on a timeline chart. Note that the lifetime of m_5 is broken from t_1 to t_2. The identifier of a message indicates its posting time, i.e., a message with a larger identifier means a newer one. Three range queries are issued to find the candidate messages for each news feed or query region. Since all candidate messages (i.e., m_1 to m_{11}) belong to the top-4 categories (i.e., restaurant, stadium, shopping, and museum), no messages are filtered out. A

Algorithm 1 n-look-ahead scheduling algorithm.

```
1: function n-LookAhead (User $u$)
2: // $u$ is at $u.location_0$ at current time $t_0$
3: // Step 1: Candidate Message Step
4: Generate $n$ locations $u.location_1, u.location_2, \ldots, u.location_n$ at times $t_1, t_2, \ldots, t_n$ for $u$,
   where $t_i = t_0 + i \times u.t_d$
5: for Each $u.location_i$ at time $t_i$ do
6:     Generate a query region $q_i$ centered at $u.location_i$ with a range distance $u.D$
7:     if $i = 0$ then
8:         Find the region of $V_u$ and compute $V_u$
9:         Use $V_u$ to compute $CM_i$
10:    else
11:        Compute new $V'_u$ based on $V_u$
12:        Use $V'_u$ to compute $CM_i$
13:    end if
14:    Filter out all candidate messages that do not belong to any $u$'s top-$\delta$ categories from $CM_i$
15:    Compute a relevance score of each message in $CM_i$
16:    Sort the messages in $CM_i$ by their relevance score in non-increasing order
17: end for
18: // Step 2: Online Scheduling Step
19: For each $q_i$, initialize an empty news feed result list $RL_i$
20: while Some $RL_i$ is not full and some $CM_i$ is not empty do
21:    For each $q_i$, calculate a score for its first candidate message $m_j$ by $relevanceScore(u, m_j) \times$
       $displayWeight(j, k)$
22:    $BestMsg \leftarrow$ the candidate message with the highest score
23:    Append $BestMsg$ to $RL_h$ giving the highest score
24:    Remove $BestMsg$ from all the candidate message sets
25:    if $RL_h$ contains $u.k$ messages then
26:        Discard all messages in $CM_h$
27:    end if
28: end while
29: Send all result lists $RL_i$ to $u$
30: // Step 3: Incremental Processing Step
31: Message Update: When a new message will improve the overall relevance score of a query region
    $q_i$, it is sent to $u$ to replace the last message in $RL_i$.
32: Inaccurate Location Prediction: When receiving a location update due to inaccurate location
    prediction, this algorithm computes a new set of news feeds. Then, only a set of messages which
    are not in the previous set of news feeds, positive and negative updates are sent to $u$.
```

relevance score is computed for each candidate message by Equation 3, as shown beside each message in the timeline chart. For each query region, its candidate messages are sorted by their relevance score, as shown in the first row in Table 2 (i.e., CM_0, CM_1, and CM_2).

Step 2. Online scheduling step. As depicted in the running example (Fig. 3), a geo-tagged message may be included in multiple sets of candidate messages. For example, m_1 is included in all three sets of candidate messages (i.e., CM_0, CM_1 and CM_2), so m_1 can be scheduled to one of the news feeds at t_0, t_1 and t_2, or none of them. This step aims at scheduling at most $k \times (n+1)$ candidate messages to the $n+1$ news feeds such that the overall relevance score of these news feeds is maximized. Here we first describe a heuristic algorithm for this step, and an optimal algorithm will be illustrated later in Section 6. For each query region q_i, we calculate a score for its first candidate message (i.e., the message with the highest relevance score) m_j by $relevanceScore(u, m_j) \times displayWeight(j, k)$ (see Equation 1), where u is the querying user and j is the highest available position in q_i's news feed result list. The message with the largest such score, denoted as *BestMsg*, is assigned to the result list of corresponding query region. Then, the scheduler removes *BestMsg* from all the candidate message sets. Candidate messages are iteratively selected to appropriate query regions until the news feed in each query region has k messages or all candidate message sets become empty. Whenever k messages have been assigned to the result list of a query region, its corresponding candidate message set is discarded. The computed $n+1$ news feeds are sent to u. The pseudo code of this step is shown in Lines 19 to 29 in Algorithm 1.

Example. Table 2 illustrates the heuristic news feed scheduler for our running example, where $n = 2$ and $k = 2$. In the first iteration, m_2 is *BestMsg* and it should be assigned to the result list of q_1 to obtain the highest score $\mathit{relevanceScore}(u, m_2) \times \mathit{displayWeight}(j, k)$. Since all three candidate message sets contain m_2, m_2 is removed from them. In the second iteration, m_1 is the next *BestMsg* and it is assigned to RL_0. This is because assigning m_1 to RL_0 has a higher score (0.58×2) than RL_1 (0.68×1). In the third iteration, m_5 is *BestMsg*, and it is assigned to the result list of q_2. In the fourth iteration, m_{10} is *BestMsg*, and it can only be assigned to q_0. At this moment, RL_0 contains k messages, so CM_0 is discarded. In the next two iterations, m_9 and m_3 are selected and assigned to q_2 and q_1, respectively. Now all news feed result lists are full, so the online scheduling step is finished and they are sent to the user. In this example, the overall relevance score of the 2-look-ahead news feeds is: $0.58\times2+0.55\times1+0.68\times2+0.43\times1+0.53\times2+0.5\times1 = 5.06$. Without any look-ahead step (i.e., $n = 0$), the news feeds at t_0, t_1, and t_2 are $\{(m_2, 0.58), (m_1, 0.58)\}$, $\{(m_3, 0.43), (m_4, 0.4)\}$, and $\{(m_5, 0.55), (m_9, 0.5)\}$, respectively, so the overall relevance score of three zero-look-ahead news feeds is: $0.58\times2+0.58\times1+0.43\times2+0.4\times1+0.53\times2+0.5\times1 = 4.56$. We can see that the improvement of 2-look-ahead scheme using the heuristic algorithm is about 11%.

Step 3. Incremental processing step. After sending the news feeds to u, MobiFeed temporarily stores the $n+1$ result lists until u's next location update. Two situations will trigger this step to maintain the news feeds:

- *Message Update.* The first situation is detected on the server side for new messages. When MobiFeed finds a new candidate message m that will improve the overall relevance score for a u's future query region q, it sends m to u to replace the last message in q's result list (Line 31 in Algorithm 1).
- *Inaccurate Location Prediction.* The second situation is detected on the client side for an inaccurate predicted location. A user's location detected by GPS always incurs some error due to the problems of measurement imprecision and sampling imprecision [18]. A tolerance threshold θ is thus defined for the difference between a user's actual location and its corresponding predicted location for a query region. If the location deviation is larger than θ, a prediction error occurs, and u reports its actual location to the server to retrieve a new set of news feeds. For example, given $n = 2$ (i.e., 2-look-ahead), for a trajectory with a user's locations at times t_0, $t_1 = t_0 + t_d$, $t_2 = t_0 + 2 \times t_d$, $\ldots$, where t_d is the minimum display time, a first call of 2-look-ahead scheduling at t_0 generates news feeds for t_0, t_1, and t_2. If no prediction error occurs from t_0 to t_2, a second call of scheduling will take place at t_3. In contrast, if prediction error occurs at t_1, our algorithm will perform re-scheduling at t_1 and generate news feeds for t_1, t_2, and t_3. To reduce communication overhead, we determine (i) a set of new messages that are in a newly computed set of news feeds, but not in a previous set of news feeds, and (ii) positive or negative updates indicate that a certain message should be added to or removed from the previous result lists, respectively [46]. MobiFeed only sends the set of new messages, positive and negative updates to the user. In practice, θ can be set to the accuracy of the

Table 2: Example of the heuristic scheduling algorithm ($n = 2$).

Iteration	***BestMsg***	**Result Lists** ($k = 2$)	**Candidate Message Sets**
Initial	-	$RL_0 = \{\emptyset\}$ $RL_1 = \{\emptyset\}$ $RL_2 = \{\emptyset\}$	$CM_0 = \{(m_2, 0.58), (m_1, 0.58), (m_{10}, 0.55), (m_5, 0.53)$ $(m_{11}, 0.5), (m_3, 0.33)\}$ $CM_1 = \{(m_2, 0.68), (m_1, 0.68), (m_3, 0.43), (m_4, 0.4)\}$ $CM_2 = \{(m_5, 0.53), (m_9, 0.5), (m_8, 0.5), (m_7, 0.4),$ $(m_2, 0.35), (m_1, 0.35), (m_6, 0.3), (m_4, 0.15), (m_3, 0.1)\}$
1	$(m_2, 0.68)$	$RL_0 = \{\emptyset\}$ $RL_1 = \{(m_2, 0.68)\}$ $RL_2 = \{\emptyset\}$	$CM_0 = \{(m_1, 0.58), (m_{10}, 0.55), (m_5, 0.53), (m_{11}, 0.5),$ $(m_3, 0.33)\}$ $CM_1 = \{(m_1, 0.68), (m_3, 0.43), (m_4, 0.4)\}$ $CM_2 = \{(m_5, 0.53), (m_9, 0.5), (m_8, 0.5), (m_7, 0.4),$ $(m_1, 0.35), (m_6, 0.3), (m_4, 0.15), (m_3, 0.1)\}$
2	$(m_1, 0.58)$	$RL_0 = \{(m_1, 0.58)\}$ $RL_1 = \{(m_2, 0.68)\}$ $RL_2 = \{\emptyset\}$	$CM_0 = \{(m_{10}, 0.55), (m_5, 0.53), (m_{11}, 0.5), (m_3, 0.33)\}$ $CM_1 = \{(m_3, 0.43), (m_4, 0.4)\}$ $CM_2 = \{(m_5, 0.53), (m_9, 0.5), (m_8, 0.5), (m_7, 0.4),$ $(m_6, 0.3), (m_4, 0.15), (m_3, 0.1)\}$
3	$(m_5, 0.53)$	$RL_0 = \{(m_1, 0.58)\}$ $RL_1 = \{(m_2, 0.68)\}$ $RL_2 = \{(m_5, 0.53)\}$	$CM_0 = \{(m_{10}, 0.55), (m_{11}, 0.5), (m_3, 0.33)\}$ $CM_1 = \{(m_3, 0.43), (m_4, 0.4)\}$ $CM_2 = \{(m_9, 0.5), (m_8, 0.5), (m_7, 0.4), (m_6, 0.3),$ $(m_4, 0.15), (m_3, 0.1)\}$
4	$(m_{10}, 0.55)$	$RL_0 = \{(m_1, 0.58),$ $(m_{10}, 0.55)\}$ $RL_1 = \{(m_2, 0.68)\}$ $RL_2 = \{(m_5, 0.53)\}$	$CM_1 = \{(m_3, 0.43), (m_4, 0.4)\}$ $CM_2 = \{(m_9, 0.5), (m_8, 0.5), (m_7, 0.4), (m_6, 0.3),$ $(m_4, 0.15), (m_3, 0.1)\}$
5	$(m_9, 0.5)$	$RL_0 = \{(m_1, 0.58),$ $(m_{10}, 0.55)\}$ $RL_1 = \{(m_2, 0.68)\}$ $RL_2 = \{(m_5, 0.53),$ $(m_9, 0.5)\}$	$CM_1 = \{(m_3, 0.43), (m_4, 0.4)\}$
6	$(m_3, 0.43)$	$RL_0 = \{(m_1, 0.58),$ $(m_{10}, 0.55)\}$ $RL_1 = \{(m_2, 0.68),$ $(m_3, 0.43)\}$ $RL_2 = \{(m_5, 0.53),$ $(m_9, 0.5)\}$	

underlying positioning technique and device. For example, if Assisted-GPS is used, θ is set to 50 meters [22] (Line 32 in Algorithm 1).

6 An Optimal News Feed Scheduler

The online scheduling step of the heuristic news feed scheduler (Step 2 in Algorithm 1) aims at scheduling at most $k \times (n+1)$ candidate messages to $n+1$ news feeds in order to maximize the total relevance score of all these news feeds. In this section, we optimize the online scheduling step as finding the maximum weight matching [55] in a weighted bipartite graph.

6.1 Maximum Weight Matching for MobiFeed

Given all sets of candidate messages CM_i $(0 \leq i \leq n)$ as the input of the online scheduling step, we form a weighted bipartite graph $G = (X \cup Y, E)$ [55] as follows:

- X is a partition of vertices that represent all the positions of $n+1$ news feeds. Thus, there are $k \times (n+1)$ vertices in X, where k is the news feed size. For vertex $x_a \in X$ where $a = k \times i + j$, it stands for the j-th position $(1 \leq j \leq k)$ in

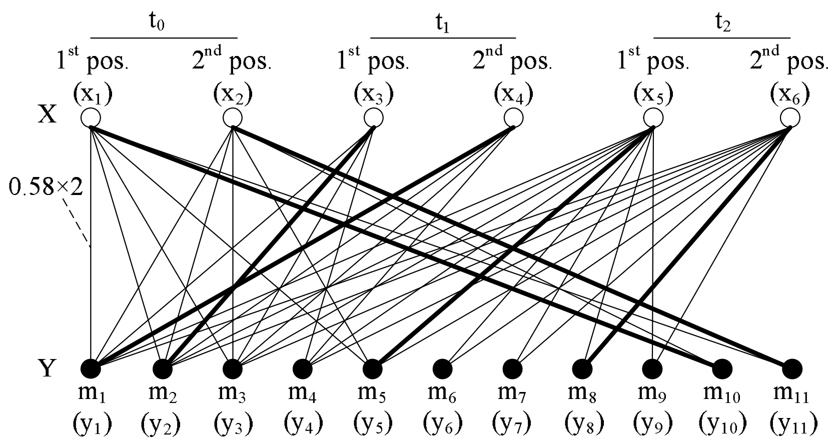


Fig. 6: The bipartite graph of our running example.

the news feed result list at t_i ($0 \leq i \leq n$). Fig. 6 depicts the bipartite graph for our running example (Fig. 3), where X contains 6 positions (i.e., two positions for each of three news feeds).

- Y is the other partition of vertices that represent all *distinct* messages in CM_i ($0 \leq i \leq n$). In our example, Y contains 11 messages (i.e., m_1 to m_{11}).
- E is a set of edges in G, for j-th position ($1 \leq j \leq k$) in the news feed at t_i ($0 \leq i \leq n$), i.e., the vertex $x_a \in X$, it links to all the messages in CM_i. For an edge (x_a, y_b), its weight $w(x_a, y_b)$ equals $relevanceScore(u, m_b) \times displayWeight(j, k)$ (see Equation 1). In our example, for the first position in the news feed at t_0 (i.e., x_1), it connects to the vertices in y_1, y_2, y_3, y_5, y_{10}, and y_{11}. And the weight of edge (x_1, y_1) equals $0.58 \times 2 = 1.16$.

As described in Section 5, a solution of the online scheduling step is defined as follows:

Definition 1 (Solution of the Online Scheduling Step) *Given $n+1$ sets of candidate messages, a solution of the online scheduling step contains $n+1$ news feeds for u's query regions at t_0, t_1, ..., and t_n, with each news feed containing at most k messages. After a solution is found, there is no available position to which an unselected candidate message can be assigned.*

Fig. 7a shows an example of input for the online scheduling step, where $n = 1$, $k = 1$. Each bold edge (t_i, m_j) in Fig. 7b, 7c, and 7d means that we assign m_j to the only position in the news feed at t_i. By Definition 1, Fig. 7b and 7c depict two solutions for this example. However, just scheduling m_1 to the news feed at t_1 (Fig. 7d) is *not* a solution, since unselected message m_2 can be assigned to an available position in the news feed at t_0.

In the modeled bipartite graph, we have matchings and maximal matchings (Definition 2 and 3). By Definition 1, each maximal matching is a solution of the online scheduling step. In the example of Fig. 7, the bold edges in Fig. 7b and 7c are maximal matchings, so they are solutions of the online scheduling step. By contrast, the matching in Fig. 7d is not maximal.

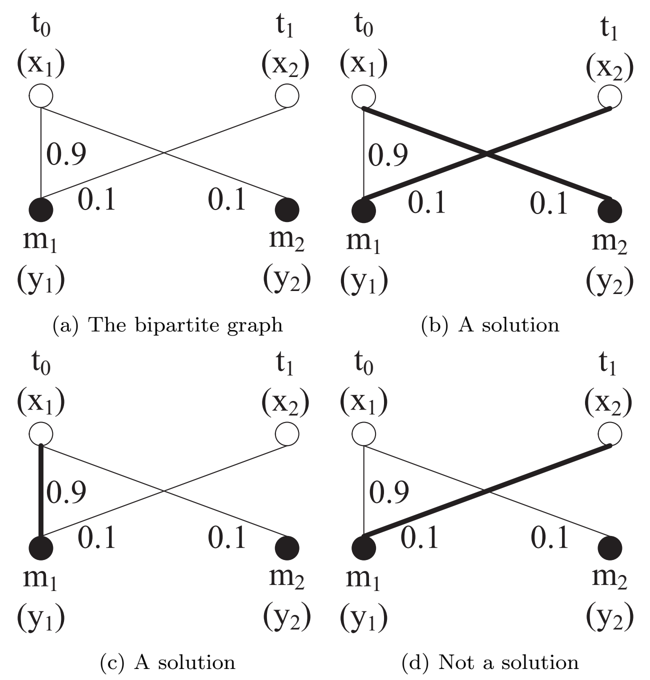


Fig. 7: Example of solutions of the online scheduling step ($k = 1$, $n = 1$).

Definition 2 (Matching) *A set of edges in a bipartite graph G is called a matching if no two edges share a common end vertex.*

Definition 3 (Maximal Matching) *A maximal matching in a bipartite graph G is a matching that cannot be enlarged by adding an edge.*

We turn to derive the optimal solution of our online scheduling step. First of all, we define the *weight* of a matching M as the sum of the weights of all edges in M, i.e., $w(M) = \sum_{e \in M} w(e)$. We then introduce the concept of maximum weight matching:

Definition 4 (Maximum Weight Matching) *The maximum weight matching in a bipartite graph G is a matching with maximum weight among all maximal matchings in G.*

By Definition 4 and the modeling of bipartite graph G, the maximum weight matching in G corresponds to the assignment of $n+1$ news feeds with the highest total relevance score, and thus is exactly the *optimal* solution of the online scheduling step. Based on this problem formulation, we describe the optimal scheduling algorithm in the next section.

6.2 Optimal Online Scheduling Algorithm

In this section, we propose an optimal algorithm for our n-look-ahead news feed scheduler. This algorithm is based on the Hungarian Algorithm [36], which finds

Algorithm 2 Optimal Online Scheduling Algorithm

```
1: function OptimalAlgorithm (User u)
2: // Step 1: Candidate Message Step (Lines 4 to 17 except Line 16 in Algorithm 1)
3: // Step 2: Online Scheduling Step
4: // Step 2a: Preprocessing
5: Construct the weighted bipartite graph $G = (X \cup Y, E)$ (see Section 6.1)
6: Scale up the weights of edges in $G$ to integers
7: Insert missing edges to make $G$ a complete bipartite graph
8: // Step 2b: Maximum weight matching
9: Initialize a label $l(v)$ for each vertex $v$ in $G$ and construct the equality subgraph $G_l$ accordingly
10: Generate an initial matching $M$ in $G_l$
11: while $|M| \neq \min(|X|, |Y|)$ do
12:     $x_s \leftarrow$ an unmatched vertex in $X$, $S \leftarrow \{x_s\}$, $T \leftarrow \emptyset$
13:     if $N_l(S) = T$ then
14:         $d_l \leftarrow \min_{x_a \in S, y_b \in Y \setminus T}\{l(x_a) + l(y_b) - w(x_a, y_b)\}$
15:         Update the vertex labels:
                $l(x_a) = l(x_a) - d_l$ for $x_a \in S$; $l(y_b) = l(y_b) + d_l$ for $y_b \in T$
16:     end if
17:     $y_t \leftarrow$ a vertex in $N_l(S) \setminus T$
18:     if $y_t$ is matched, say to $x'$, then
19:         $S \leftarrow S \cup \{x'\}$, $T \leftarrow T \cup \{y_t\}$, go to Line 13
20:     else
21:         There is an augmenting path $P$ in $G_l$ from $x_s$ to $y_t$. $M \leftarrow M \ominus P$
22:     end if
23: end while
24: // Step 2c: Assignment
25: For each edge $(x_a, y_b) \in M$, if its weight is larger than zero, assign message $m_b$ to the position
    represented by $x_a$ (see Section 6.1).
26: Send generated news feeds to $u$
27: // Step 3: Incremental Processing Step (Lines 31 and 32 in Algorithm 1)
```

the maximum weight matching in a bipartite graph G with two *equal-size* partitions. However, in our news feed scheduling problem, two partitions in G usually do not have the same number of vertices (see Section 6.1). Thus, we modify the Hungarian Algorithm such that it could be applied to our problem. Finally, we prove the correctness of our optimal algorithm.

Algorithm. As shown in Algorithm 2, the Steps 1 and 3 of the optimal online scheduling algorithm are almost the same as in Algorithm 1, except that we do not require candidate messages CM_i ($0 \leq i \leq n$) to be sorted (Line 16 in Algorithm 1). We focus on the online scheduling step (Step 2), which can be further divided into three steps.

Step 2a. Preprocessing. With $n + 1$ sets of candidate messages generated by Step 1, we construct the weighted bipartite graph $G = (X \cup Y, E)$ as described in Section 6.1. Since the Hungarian Algorithm works on graphs with integer-weight edges [6], we scale up the weights of edges in G to integers. Furthermore, the Hungarian Algorithm requires that G should be a complete bipartite graph (i.e., every vertex of one partition is connected to every vertex of the other partition) [6], so we insert the missing edges with zero weights. This does not affect the weight of a matching we can obtain.

Step 2b. Maximum weight matching. This step aims at finding the maximum weight matching in G. To achieve this, we assign a label $l(v)$ (i.e., an integer value) for each vertex $v \in X \cup Y$, which is initialized as follows [6]: $l(x_a) = \max\{w(x_a, y_b) | y_b \in Y\}$ for $x_a \in X$; $l(y_b) = 0$ for $y_b \in Y$. The labels will be changed by the algorithm and they must satisfy $l(x_a) + l(y_b) \geq w(x_a, y_b)$ for each $x_a \in X$, $y_b \in Y$. Then, we construct the *equality subgraph* G_l with respect to

$$\begin{array}{cccccccccccc}
 & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 & y_9 & y_{10} & y_{11} \\
x_1 & \overset{0.58\times2\times100}{116} & 116 & 66 & 0 & 106 & 0 & 0 & 0 & 0 & 110 & 100 \\
x_2 & 58 & 58 & 33 & 0 & 53 & 0 & 0 & 0 & 0 & 55 & 50 \\
x_3 & 136 & 136 & 86 & 80 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x_4 & 68 & 68 & 43 & 40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x_5 & 70 & 70 & 20 & 30 & 106 & 60 & 80 & 100 & 100 & 0 & 0 \\
x_6 & 35 & 35 & 10 & 15 & 53 & 30 & 40 & 50 & 50 & 0 & 0
\end{array}$$

Fig. 8: Weight matrix of the bipartite graph.

the labels l. More specifically, G_l contains all vertices in G, as well as edges (x_a, y_b) in G which satisfy $l(x_a) + l(y_b) = w(x_a, y_b)$. By Definition 2, we can generate an initial matching M in G_l by checking the neighbors of every vertex in X. The size of M is increased by one in each iteration (Lines 12 to 22 in Algorithm 2) until $|M| = \min(|X|, |Y|)$. When the loop terminates, the matching M in G_l is the maximum weight matching in G. Note that the termination condition of the loop (i.e., Line 11) is different from that in the Hungarian Algorithm (i.e., $|M| = |X| = |Y|$).

To increase the size of matching M, the idea is to add new edges in G_l by adjusting the vertex labels l, and then find an *augmenting path* P [6] in G_l, which is a path with (a) edges alternating between M and $\overline{M}$ (called an *alternating path* [6]), and (b) unmatched start and end vertices. Specifically, in each iteration, we pick an unmatched vertex x_s in X, and initialize a set S with $\{x_s\}$ and an empty set T. We find an augmenting path by exploring alternating paths starting from x_s. Basically, S and T are used to record vertices that have been explored in X and Y, respectively. We then determine the *neighbors* of S (denoted as $N_l(S)$), i.e., the vertices in Y that can be reached (in terms of G_l) from any vertex in S. When $N_l(S) = T$, all alternating paths from x_s to vertices in Y have been explored but no augmenting path is found, then we need to add new edges into the equality subgraph G_l. This is achieved by updating the vertex labels l, as shown in Lines 14 and 15 in Algorithm 2. This update operation guarantees that new edges incident to vertices in S are added into G_l [6], which leads to $N_l(S) \neq T$, so we can find a vertex $y_t \in N_l(S) \setminus T$. If y_t is matched, say to vertex x' in X, then x' and y_t are inserted into S and T, respectively, and the algorithm turns back to check the equality between $N_l(S)$ and T again (Line 19). Otherwise, there exists an augmenting path P in G_l from x_s to y_t. Thus, M can be augmented by computing the symmetric difference of M and P, i.e., $M \ominus P$ (Line 21). This step terminates when $|M| = \min(|X|, |Y|)$.

Step 2c. Assignment. For each edge $(x_a, y_b) \in M$, if its weight is larger than zero, the corresponding message m_b is assigned to the position represented by x_a (see Section 6.1). After checking every edge in M, the $n + 1$ news feeds constitute an optimal solution for our online scheduling step.

Example. We use our running example in Fig. 6 to illustrate the three steps of the optimal online scheduling step.

Step 2a: Fig. 8 depicts the weighted bipartite graph G that is represented by a *weight matrix* with 6 rows and 11 columns. In the matrix, the value in entry (a, b) is the weight $w(x_a, y_b)$ that is scaled by 100 to keep two decimal points for relevance scores.

Step 2b: As shown in Fig. 9a, we get the equality subgraph G_l with respect to the initial vertex labels l (in square brackets). By checking the neighbors of x_1, x_2, $\ldots$, and x_6, we obtain an initial matching $M = \{(x_1, y_1), (x_2, y_2), (x_5, y_5)\}$ in G_l (bold edges in Fig. 9a) (i.e., no messages can be assigned to x_3, x_4, or x_6). In the first iteration, x_3 is selected as an unmatched vertex, and S is initialized as $\{x_3\}$. Then, y_1 is the vertex picked in $N_l(S)\backslash T$. We add y_1 and its matched vertex x_1 into T and S, respectively. Since $N_l(S)$ is still not equal to T, y_2 is inserted into T. After its matched vertex x_2 is added into S, $N_l(S)$ is equal to T. We calculate $d_l = 3$ according to Line 15 in Algorithm 2, and then decrease the labels in S (i.e., x_1, x_2, and x_3) by $d_l = 3$ and increase the labels in T (i.e., y_1 and y_2) by $d_l = 3$ (Fig. 9b). Now a new edge (x_2, y_{10}) is added into the equality subgraph, since $l(x_2)+l(y_{10}) = w(x_2, y_{10})$ after updating the vertex labels. When the unmatched vertex y_{10} is explored as a neighbor of x_2, an augmenting path P from x_3 to y_{10} can be obtained as $x_3 \rightarrow y_2 \rightarrow x_2 \rightarrow y_{10}$ (highlighted in Fig. 9b). After augmenting M by $M \ominus P$, we obtain an expanded matching $M = \{(x_1, y_1), (x_2, y_{10}), (x_3, y_2), (x_5, y_5)\}$ (Fig. 9c). This step repeats such iterations until $|M| = \min(|X|, |Y|) = 6$. The final matching is $M = \{(x_1, y_{10}), (x_2, y_{11}), (x_3, y_2), (x_4, y_1), (x_5, y_5), (x_6, y_8)\}$, represented by bold edges in Fig. 6.

Step 2c: After the assignment step, the corresponding news feeds at t_0, t_1, and t_2 are $\{(m_{10}, 0.55), (m_{11}, 0.5)\}$, $\{(m_2, 0.68), (m_1, 0.68)\}$, and $\{(m_5, 0.53), (m_8, 0.5)\}$, respectively. The total relevance score of the 2-look-ahead news feeds generated by the optimal scheduler is: $0.55\times2+0.5\times1+0.68\times2+0.68\times1+0.53\times2+0.5\times1 = 5.2$. Compared with the total relevance score of the zero-look-ahead algorithm (i.e., 4.56, as calculated in Section 5), the improvement of the optimal 2-look-ahead algorithm is about 14%.

Correctness. We prove the correctness of our optimal online scheduling algorithm by Theorem 1.

Theorem 1 *The matching generated by Step 2b of Algorithm 2 is the maximum weight matching in G.*

Proof Case 1: $|X| < |Y|$. We first prove that after the main loop (Lines 11 to 23 in Algorithm 2) terminates, the labels for unmatched vertices in Y are all *zero*. Firstly, before entering the loop, the initial labels for vertices in Y are zero. Secondly, in the loop, according to the update rule of vertex labels (Lines 14 and 15 in Algorithm 2), the label for any vertex in Y is only updated if it belongs to set T, and all vertices in T are matched vertices (Line 19). Consequently, the labels for unmatched vertices in Y remain unchanged as zero when the loop terminates.

We prove that the matching M in G_l generated by Step 2b is the maximum weight matching in G. Since G_l is a subgraph of G, M is a matching in G. Also, due to the fact that (1) $|M| = |X|$, i.e., each vertex in X and each matched vertex in Y are covered only once by M, (2) the labels for unmatched vertices in Y are 0 (as discussed above), and (3) G_l is the equality subgraph derived from l, we have: (a) $w(M) = \sum_{e\in M} w(e) = \sum_{v\in X\cup Y} l(v)$. On the other hand, assume that M' is any matching in G. For any edge $(x_a, y_b) \in M'$, we have: $l(x_a) + l(y_b) \geq w(x_a, y_b)$ (i.e.,

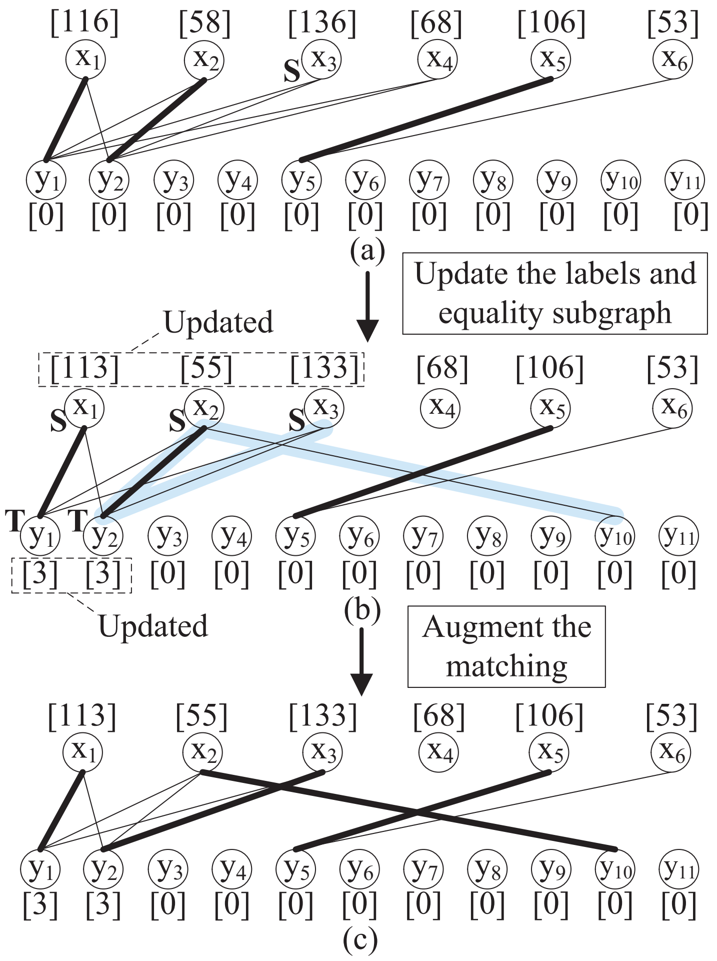


Fig. 9: Example of maximum weight matching.

the property of vertex labels l), and hence, (b) $w(M') \leq \sum_{v \in X \cup Y} l(v)$. Combining inequalities (a) and (b), $w(M) \geq w(M')$. Therefore, M is the maximum weight matching in G.

Case 2: $|X| > |Y|$. This case is symmetric to Case 1.

Case 3: $|X| = |Y|$. The original Hungarian algorithm works exactly on this case [6].

6.3 Discussion

The "optimality" of our scheduling algorithm is defined as follows: given a user's location at the query time and n predicted locations, our algorithm generates a news feed for each of these $n+1$ locations, such that the total relevance score of these news feed is *maximized*.

7 Complexity Analysis

In this section, we analyze the time complexity of the heuristic news feed scheduler (Algorithm 1) and the optimal news feed scheduler (Algorithm 2). As defined in Section 6, let X denote all available positions in $n+1$ news feeds, i.e., $|X| = k \times (n+1)$, and let Y represent the set of *distinct* messages in all CM_i $(0 \le i \le n)$. Since the complexity of the Step 3 of the heuristic and optimal schedulers is the same, we focus on their first two steps.

Heuristic news feed scheduler. In Algorithm 1, Step 1 takes $O((n{+}1)|Y|\log|Y|)$ to sort candidate messages. Step 2 takes constant time to schedule each *BestMsg*. Since the goal is to schedule at most $k(n+1)$ messages, the complexity of Step 2 is $O(k(n{+}1))$. As a result, the complexity of the heuristic scheduler is $O(n(|Y|\log|Y|+k))$.

Optimal news feed scheduler. In Algorithm 2, Step 1 only takes $O((n{+}1)|Y|) = O(|X||Y|)$ (because $|X| = k \times (n+1)$) to retrieve all candidate messages. Step 2a takes $O(|X||Y|)$ time to construct G. We can distinguish two cases in Step 2b:

Case 1: $|X| \le |Y|$. There are at most $|X|$ iterations in Step 2b. In each iteration, at most $|X|$ vertices can be moved from $\bar{S}$ to S. For each of such operations, we may update the vertex labels l. Just using formulas in Lines 14 and 15 directly will lead to a cost of $O(|X||Y|)$. For the sake of efficiency, we use a *slack* array [55] initialized in $O(|Y|)$ time (S only contains one element initially):

$$\mathit{slack}[y_b] = \min_{x_a \in S}(l(x_a) + l(y_b) - w(x_a, y_b)), y_b \in Y$$

When one vertex is moved from $\bar{S}$ to S, it takes $O(|Y|)$ time to update the *slack* array. The calculation of d_l takes $O(|Y|)$ time, since $d_l = \min_{y_b \in Y \setminus T} \mathit{slack}[y_b]$, and additional $O(|Y|)$ time is needed to update the labels l. Thus, the label update operation needs $O(|Y|)$ time, and the complexity of Step 2b is $O(|X|^2|Y|)$.

Case 2: $|X| > |Y|$. Similarly, in this case, the complexity of Step 2b is $O(|X||Y|^2)$. Step 2c takes $O(\min(|X|, |Y|))$ to assign every edge in M. As a result, the complexity of the optimal scheduler is $O(|X|^2|Y|)$ or $O(|X||Y|^2)$ for Case 1 or 2, respectively.

8 Experiment Results

In this section, we evaluate the performance of MobiFeed through experiments using a real location-aware social networking data set collected from FourSquare (Sections 8.2, 8.3, and 8.4). Also, we study the scalability of MobiFeed using a synthetic data set (Section 8.5). We first describe our experiment settings, and then analyze experiment results.

8.1 Experiment Setting

Road map and data sets. We extracted the road map of NYC from the USA Census TIGER/Line Shapefiles [1] and randomly generated 5,000 30-minute trajectories using the A* algorithm [29]. In the first three sets of experiments (Sections 8.2, 8.3, and 8.4), we use a real location-aware social network data set in

Table 3: Space cost of different schedulers

Memory Usage	**GeoFeed**	**zero-LA**	**n-LA-H**	**n-LA-O**	**n-LA-H (w/o view)**	**n-LA-O (w/o view)**
Road Map	506K	506K	506K	506K	506K	506K
Grid Index	49M	49M	49M	49M	49M	49M
Bipartite Matching	N/A	N/A	N/A	124K	N/A	124K
Materialized View	N/A	N/A	84K	84K	N/A	N/A
Others	6K	8K	9K	12K	9K	12K

New York City (NYC), USA, which was crawled from FourSquare [25] from December 10 to 25, 2012. The data set contains 417,897 geo-tagged user-generated messages (belong to 420 categories in total) and 101,232 users. We also use a synthetic data set to evaluate the scalability of MobiFeed in Section 8.5.

Parameters and performance measure. Unless mentioned otherwise, all users move at a constant speed of 40km/h, the minimum message display time (t_d) is 20 seconds, the look-ahead step (n) is 5, the number of messages per news feed (k) is 5, the number of top categories (δ) is 3, and the query range distance (D) is 600 meters. In Section 8.2, we first consider no location prediction error and evaluate the performance of MobiFeed with respect to various parameters. After that, we discuss the relevance measure function considering temporal factor in Section 8.3. Then, in Section 8.4 we study the effect of prediction errors of the location prediction function described in Section 4.1.

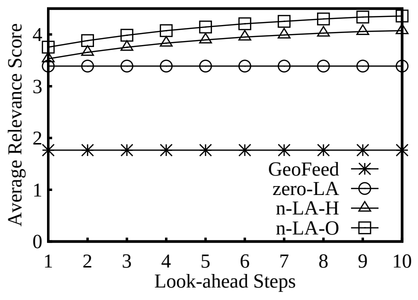
We measure the quality and efficiency of MobiFeed in terms of the average relevance score and running time per news feed, respectively. For comparison, a GeoFeed user issues a query at every t_d to retrieve the k most *recent* messages. Also, we implement the *zero-look-ahead* scheduler (termed zero-LA) as another baseline, and evaluate the performance of the heuristic *n-look-ahead* scheduler (termed n-LA-H) and the optimal *n-look-ahead* scheduler (termed n-LA-O).

All the experiments were implemented in C++ and run on an Ubuntu 13.10 machine with a 3.4GHz Intel Core i7-4770 processor and 16GB RAM. In the experiments, the road network and largest dataset occupies 1 MB and 50 MB, respectively. Thus, all data can fit in the main memory.

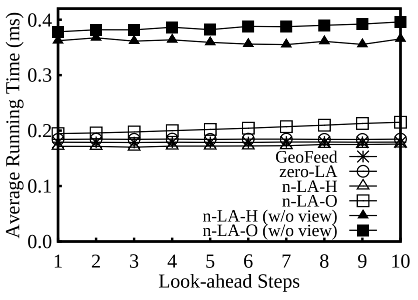
8.2 Performance Study with Various Parameters

In this section, we investigate the performance of MobiFeed by varying parameters including (a) the look-ahead step, (b) the minimum display time, (c) the new feed size, (d) the query range distance, (e) the number of top categories, and (f) the user movement speed.

Effect of the Number of Look-ahead Steps. Fig. 10 and Table 4 depict the performance of MobiFeed with respect to 1 to 10 look-ahead steps (n). The performance of zero-LA and GeoFeed is not affected by the value of n. When n is larger, both n-LA-H and n-LA-O have a higher chance to assign a message to a better news feed within its lifetime; and hence, the quality of news feeds generated by n-LA-H and n-LA-O improves (Fig. 10a). Besides, n-LA-O obtains a relevance score about 5% higher than n-LA-H and about 15% higher than zero-LA on average. Fig. 10a



(a) Relevance score



(b) Running time

Fig. 10: Look-ahead steps.

Table 4: The number of location updates.

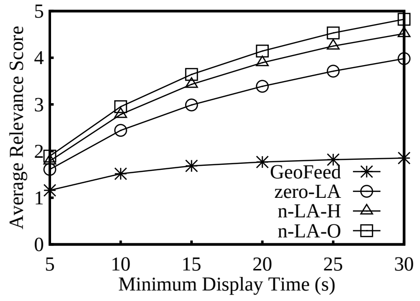
Look-ahead steps	1	2	3	4	5	6	7	8	9	10
n-LA-H/n-LA-O	90	60	45	36	30	26	23	20	18	17
zero-LA/GeoFeed	180	180	180	180	180	180	180	180	180	180

also shows that GeoFeed performs worst among all approaches. The reason is that, GeoFeed simply uses submission time as the ranking measure, while our schedulers use a more advanced ranking function (described in Section 4.2).

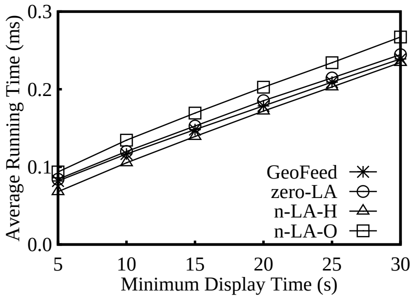
Interestingly, compared with zero-LA and GeoFeed, although n-LA-H and n-LA-O have extra overhead in assigning messages to a news feed within their lifetime, this overhead is almost offset by the benefit of sharing the computation of scheduling $n + 1$ news feeds for a user's location (Fig. 10b). Fig. 10b also depicts that materialized views help to reduce the query processing time for n-LA-H and n-LA-O. The main reason is that the overlapping area between consecutive query regions is large, and most of the messages stored in the views can be reused for the evaluation of range queries. Since in other experiments, the improvement of efficiency for using views is similar, we will not depict the results for schedulers without views in other experiments. Table 4 shows that when n increases, n-LA-H and n-LA-O are able to pre-compute more news feeds for a mobile user's location update; thus, the number of location updates reduces.

In Table 3 we show the space costs of different schedulers by breaking down their memory usage. The grid index for all messages consumes the majority of the whole space cost. The materialized view for our schedulers does not consume much memory. The reason is that, the view keeps being updated with the candidate messages in the *most recent* query region, and the quantity of these messages would not be very large.

Effect of Minimum Display Time. In this experiment, we evaluate the effect of the minimum message display time (i.e., t_d), varying from 5 to 30 seconds, on MobiFeed. Fig. 11a depicts that the quality of news feeds improves when t_d gets larger. n-LA-H and n-LA-O also perform better than zero-LA and GeoFeed as t_d gets larger. The main reason is that, as t_d gets larger, the distance between the centers of two consecutive query regions increases; thus, the overlapping area between two

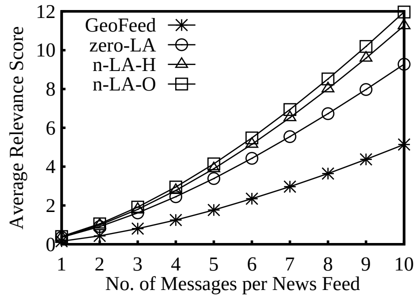


(a) Relevance score

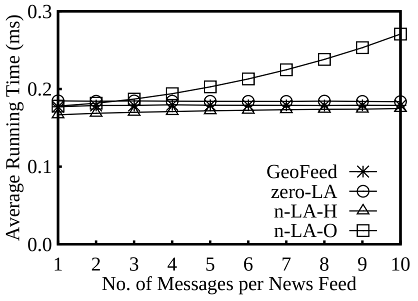


(b) Running time

Fig. 11: Minimum display time.



(a) Relevance score

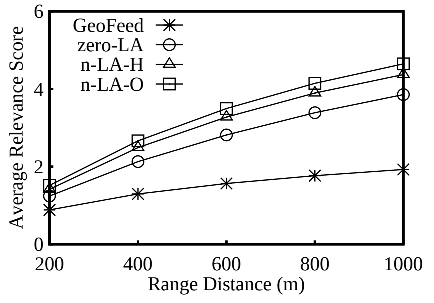


(b) Running time

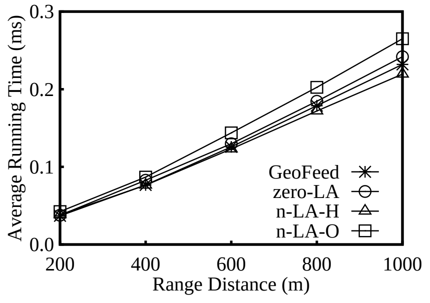
Fig. 12: News feed sizes.

consecutive query regions reduces (given fixed range distance). As a result, there are more distinct candidate messages for $n + 1$ query regions, and the schedulers have a higher chance to generate news feeds with better quality. However, t_d should not be too long because mobile users would miss many nearby relevant messages. As depicted in Fig. 11b, the running time of MobiFeed increases when t_d gets larger, since all the schedulers have to process more distinct candidate messages.

Effect of News Feed Sizes. Fig. 12 depicts the performance of MobiFeed with respect to various user-required number of messages per news feed (i.e., k). It is expected that the average relevance score of a news feed increases when a mobile user requires more messages per news feed (i.e., a larger value of k) (Fig. 12a). n-LA-O performs best since it optimizes the scheduling of news feeds over $n + 1$ query regions. Furthermore, the improvement of n-LA-H and n-LA-O over zero-LA increases when k gets larger (Fig. 12a). The reason is that when zero-LA selects more messages for a current news feed, it has a higher chance to reduce the quality of subsequent news feeds; however, some selected messages could be put in a higher position in a subsequent news feed by n-LA-H and n-LA-O. For n-LA-O, a larger k

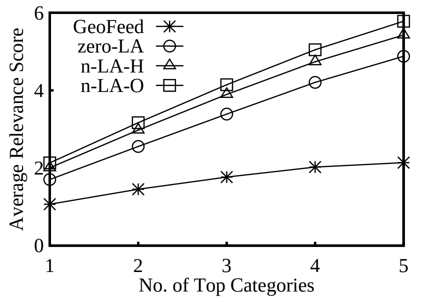


(a) Relevance score

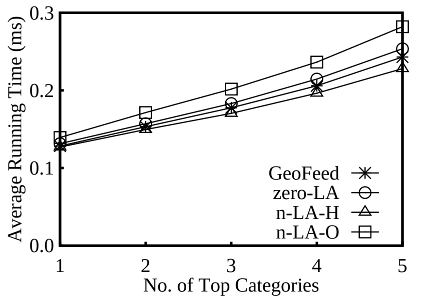


(b) Running time

Fig. 13: Query range distances.



(a) Relevance score



(b) Running time

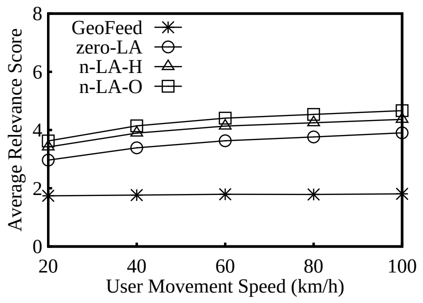
Fig. 14: The number of top categories.

needs more time to compute the maximum weight matching, so its running time increases (Fig. 12b).

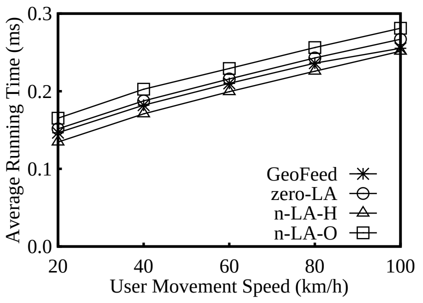
Effect of Query Range Distances. This section studies the effect of user-specified query range distances (i.e., D) on the performance of MobiFeed by varying D from 200 to 1,000 meters. The increase of D leads to more candidate messages for a query region. The news feed quality of MobiFeed improves when the number of candidate messages gets larger (Fig. 13a), since there is a higher chance to find more relevant messages for a user. However, Fig. 13b shows that the running time of all the schedulers gets worse when D increases. The main reason is that the system needs to process more candidate messages to generate news feeds.

Effect of the Number of Top Categories. Fig. 14 depicts the results of varying the number of top categories (i.e., δ) from 1 to 5. When δ increases, more candidate messages belong to the top-δ categories for a news feed, so all the schedulers have a higher chance to find news feeds with better quality (Fig. 14a). However, the increase of δ results in higher running times (Fig. 14b) because all the schedulers have to process more candidate messages for each query region.

Effect of User Movement Speeds. This experiment varies user's movement



(a) Relevance score



(b) Running time

Fig. 15: User movement speeds.

speeds from 20 to 100 km/h. A higher speed reduces the overlapping area of two consecutive query regions, so the query regions have more distinct candidate messages. As a result, there is a higher chance for our schedulers to generate news feeds with better quality (Fig. 15a). Correspondingly, since all schedulers have to process more distinct candidate messages, their running times increase as the user's speed gets higher.

8.3 Study on Relevance Measure Function with Temporal Factor

In Section 4.2, we have defined the relevance measure function by considering the message's content, associated categories, and distance to the querying user (see Equation 3). We here consider a temporal factor as an addition criterion for relevance measure. Intuitively, the interestingness of a message's content would fade as time goes, so we re-define Equation 3 as follows:

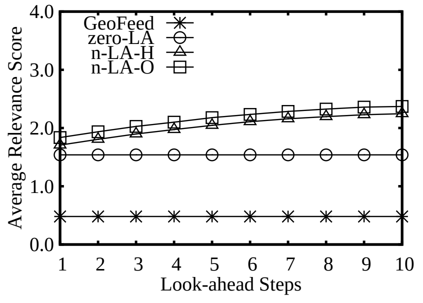
$$\begin{aligned} \mathit{relevanceScore}(u_i, m_j, T_{cur}) =& \mathit{contentScore}(u_i, m_j) \times \mathit{fading}(T_{cur}, T_{m_j}) \times (1-\beta) \\ &+ \mathit{NDist}(u_i, m_j) \times \beta \end{aligned} \tag{4}$$

where $\mathit{fading}(T_{cur}, T_{m_j})$ indicates how fast the interestingness of m_j fades; specifically, it is defined as follows:

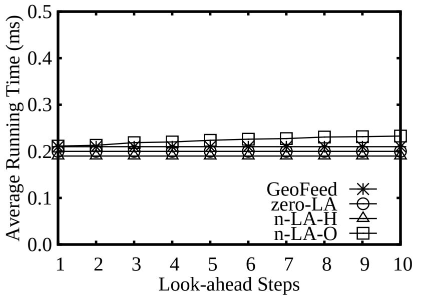
$$\mathit{fading}(T_{cur}, T_{m_j}) = 1 - \frac{T_{cur} - T_{m_j}}{T_{cur} - T_{min}} \tag{5}$$

where T_{cur}, T_{m_j} and T_{min} are the current timestamp, m_j's submission timestamp and the smallest submission timestamp for all candidate messages within current query region, respectively.

We conduct an experiment to show the performance of MobiFeed with the updated relevance measure function. The experimental results (Figure 16) exhibit similar trends compared to those with original relevance measure function (i.e., Figure 10 in Section 8.2).

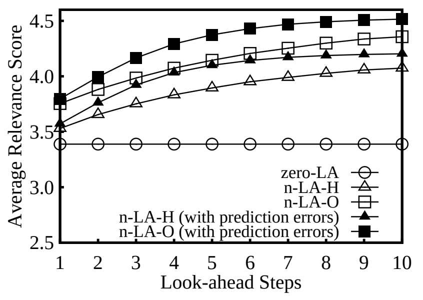


(a) Relevance score

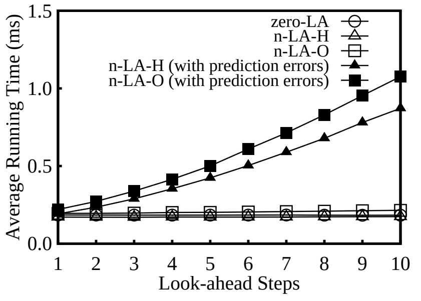


(b) Running time

Fig. 16: Performance of MobiFeed with updated relevance measure function



(a) Relevance score



(b) Running time

Fig. 17: Look-ahead steps (with prediction errors).

Table 5: Prediction error (look-ahead steps).

Look-ahead steps	1	2	3	4	5	6	7	8	9	10
Prediction error (%)	1.7	15.5	18.5	19.7	20.4	20.8	21.0	21.3	21.4	21.5

8.4 Study on Location Prediction Errors

Training set and test set. This section evaluates the effect of prediction errors of the *location prediction* function (Section 4.1). We randomly generate 10,000 trajectories on the road map with speeds randomly selected from 20km/h to users' maximum movement speed (40 km/h by default). 5,000 trajectories are randomly selected as a training set for the *location prediction* function, and $\alpha = 1$ (see Section 4.1). We use the remaining 5,000 trajectories as a test set to evaluate the effect of the prediction algorithm.

Prediction error measure. In the experiments, a *prediction error* occurs if the difference between a user's actual location and predicted location is larger than the tolerance threshold θ that is set to 50 meters. When a prediction error takes place, the scheduler needs to re-schedule news feeds based on a user's location update.

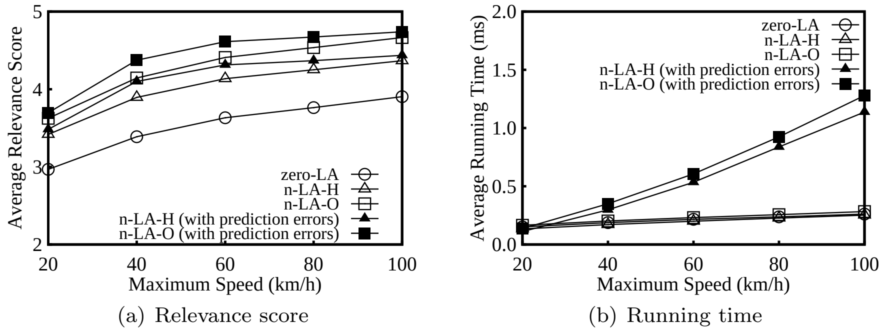


Fig. 18: Maximum user movement speeds (with prediction errors).

Table 6: Prediction error (user maximum speeds).

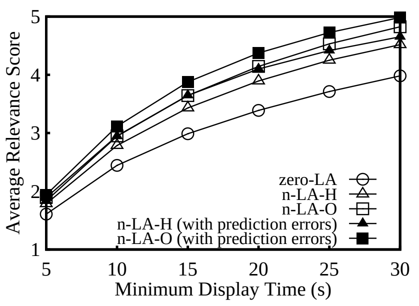
Maximum speed (km/h)	20	40	60	80	100
Prediction error (%)	3.0	20.4	38.6	50.9	59.5

We measure the prediction error of an experiment as the ratio of the number of prediction errors to the number of location predictions. In the experiments, we make a performance comparison between the n-look-ahead schedulers with prediction errors and those without errors. Although zero-LA and GeoFeed do not require location prediction, we plot the performance of zero-LA for reference.

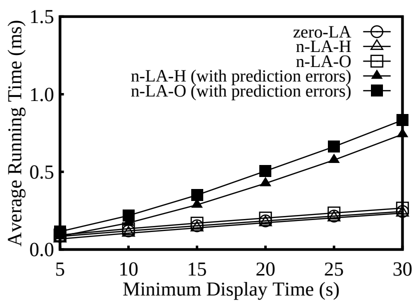
Varying look-ahead steps. Fig. 17 and Table 5 depict the experiment results of the increase of the look-ahead steps (n) from 1 to 10. Fig. 17b shows that, a larger n increases the running times of our schedulers with errors in *location prediction* function. This is because as n gets larger, MobiFeed needs to predict longer path, and there is a higher chance for a prediction error to occur (Table 5). When a prediction error takes place, the scheduler needs to re-schedule news feeds based on a user's location update, thus incurring additional computational overhead.

Interestingly, as shown in Fig. 17a, the schedulers with prediction errors generate better news feeds than those without prediction errors. The main reason is that, schedulers with errors incur additional news feed re-scheduling over those without errors, which leads to better assignment of news feeds in a *global* sense. Here we give an example for explanation. Given a trajectory with a user's locations at times t_0, $t_1 = t_0 + t_d$, $t_2 = t_0 + 2 \times t_d$, $\ldots$, where t_d is the minimum display time, a first call of 2-look-ahead scheduling at t_0 generates news feeds for t_0, t_1, and t_2. If no prediction error occurs from t_0 to t_2, a second call of scheduling will take place at t_3. In contrast, if MobiFeed detects prediction error at t_1, it will perform re-scheduling at t_1 and generate news feeds for t_1, t_2, and t_3; in this case, some messages originally assigned to news feed at t_2 may be re-assigned to news feed at t_3, which results in higher *overall* quality of news feeds from t_0 to t_3.

Varying other parameters. Since varying parameters, including k, D and δ, does not affect the prediction error, we only show the results of user's maximum movement speed (S_{max}) and the minimum display time (t_d). Fig. 18 and Table 6 depict



(a) Relevance score



(b) Running time

Fig. 19: Minimum display times (with prediction errors).

Table 7: Prediction error (minimum display times).

Minimum display time (s)	5	10	15	20	25	30
Prediction error (%)	9.5	20.4	30.2	37.8	43.9	49.1

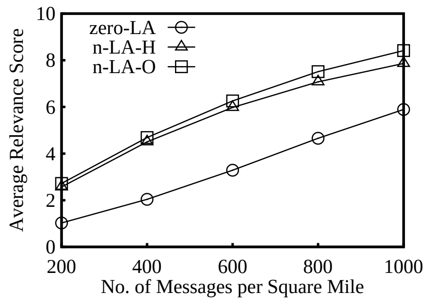
the performance of MobiFeed with respect to various user's maximum movement speeds S_{max} from 20 to 100 km/h. The minimum movement speed is 20 km/h. It is expected that the prediction error increases, as S_{max} gets higher (Table 6). The increase of the prediction error leads to a higher chance for the scheduler to re-schedule news feeds, so the running time increases (Fig. 18b). Similar to Fig. 17a, re-scheduling incurred by prediction error increases the quality of news feeds (Fig. 18a).

Fig. 19 and Table 7 depict the results of varying the minimum display time (t_d). When t_d increases, n-LA-H and n-LA-O need to predict a longer path that leads to a higher prediction error (Table 7). Similar to the previous two experiments, the increase of the prediction error results in higher running time (Fig. 19b).

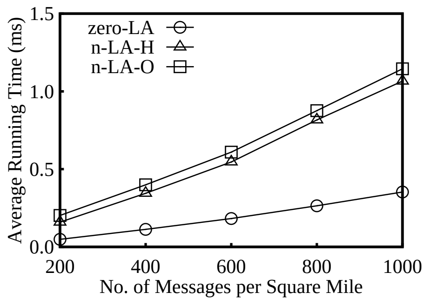
8.5 Scalability Study with Synthetic Data Set

In this experiment, we evaluate the scalability of MobiFeed using a synthetic data set. Specifically, we use the same NYC road map as previous experiments. We also consider location prediction errors by dividing trajectories into training set and test set. Instead of using real geo-tagged messages crawled from FourSquare, we generate points in the road map and regard them as geo-tagged messages. We control the features of this synthetic data set as follows:

- *Message density.* We divide the road map of NYC into 1 mile $\times$ 1 mile cells, and generate a designated number (denoted as ρ) of messages in each cell. The default value of ρ is 600. Note that in the real social network data set, there are about 250 messages per square mile.
- *Message distribution.* When generating messages *in each cell*, we control the distribution of messages. Specifically, for a distribution type 'Gau[σ%]', we regard a cell as a Gaussian bell, set the *standard deviation* of the Gaussian



(a) Relevance score



(b) Running time

Fig. 20: Message density.

to be σ% of the cell domain length[1] [10], and generate messages in that bell following such distribution. We name σ as the *percentile standard deviation.* Smaller σ means that the geo-tagged messages are clustered around the center of the cell to a larger extent. Note that when σ is set to 100 (the default value), the messages follow a uniform distribution.

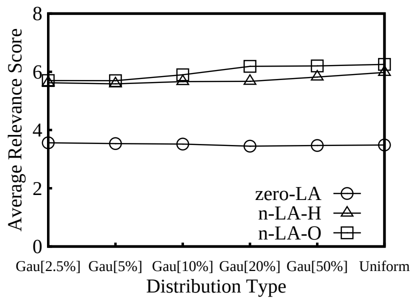
- *Score distribution.* Instead of using the ranking function (i.e., scoring function) described in Section 4.2, we generate the scores of synthetic messages following a Zipfian distribution [28]. Such distribution is determined by the skew parameter θ ranging from 0.0 to 1.0, where a smaller value of θ results in higher skew of message scores (i.e., more low-scored messages). Note that when θ is set to 1.0 (the default value), the scores follow a uniform distribution.

In the following experiments, we test the performance of MobiFeed by varying these three features of the synthetic data set. Other parameters of MobiFeed take their default values as described in Section 8.1.

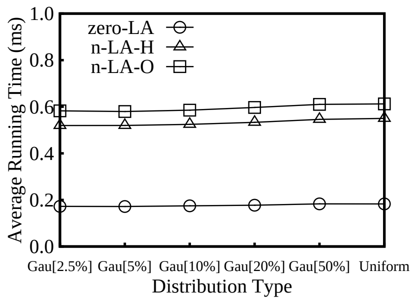
Effect of message density. Fig. 20 depicts the experiment results by varying the number of messages per square mile (i.e., ρ). The news feed quality of MobiFeed improves as the geo-tagged messages become denser (Fig. 20a), since there are more candidate messages for a query region, and the schedulers have a higher chance to find more relevant messages for a user. However, Fig. 20b shows that the running time of all schedulers gets worse as there are more messages per square mile, since the system needs to process more candidate messages to generate news feeds. As a remark, even when ρ increases to 1000, the running time of MobiFeed is still acceptable (about 1ms).

Effect of message distribution. This experiment illustrates the effect of message distribution on the performance of the MobiFeed. Fig. 21a shows that, if the geo-tagged messages are uniformly distributed, the performance gap between n-LA-O and n-LA-H is not that evident. When the messages follow the Gaussian distribution with percentile standard deviation (i.e., σ) equal to 20, n-LA-O shows greatest advantage over n-LA-H in terms of the quality of generated news feeds. Fortunately,

[1] Here the cell domain length is set as the largest distance between the cell's center and its boundary.

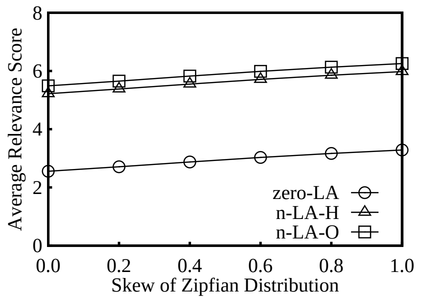


(a) Relevance score

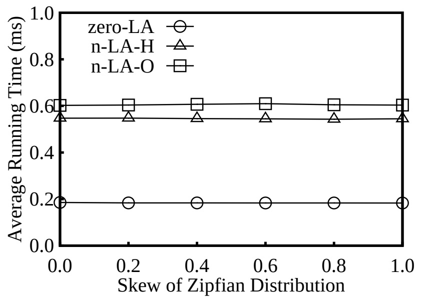


(b) Running time

Fig. 21: Message distribution.



(a) Relevance score



(b) Running time

Fig. 22: Score distribution.

in real location-aware social networking systems, the geo-tagged messages do exhibit a geographical clustering phenomenon [59], rendering our n-LA-O superior to the heuristic approach. However, if σ further gets smaller, their performance gap decreases. The reason is that, as messages further cluster around the center of cells, there is a higher chance that the number of candidate messages in a query region is less than k, rendering the performance of n-LA-O and n-LA-H almost the same. Fig. 21b shows that the running time of all schedulers is insensitive to the distribution of messages.

Effect of score distribution. This experiment varies the skewness of message score distributions. Fig. 22a shows that the quality of news feeds improves as the score becomes less skewed. The reason is that, as mentioned in the synthetic data generation, lower skewness indicates larger population of high-scored messages, and there is a higher chance for schedulers to generate news feeds with better quality. As depicted in Fig. 22b, the score distribution only slightly affects the efficiency of our schedulers.

9 Conclusion

In this paper, we presented MobiFeed that is a framework designed for scheduling location-aware news feeds for mobile users. MobiFeed has three key functions: *location prediction*, *relevance measure*, and *news feed scheduler*. We proposed the n-look-ahead *heuristic* and *optimal news feed schedulers* that work with the other two functions to generate news feeds for a user at her current and n future predicted locations. We evaluated the performance of MobiFeed through extensive experiments using a real road map, a real location-aware social network data set and a synthetic data set. The experiment results show that MobiFeed provides efficient and high-quality news feeds for mobile users.

Acknowledgments

Wenjian Xu and Chi-Yin Chow were supported by Guangdong Natural Science Foundation of China under Grant S2013010012363 and a research grant (CityU Project No. 9680117). Man Lung Yiu was supported by grant GRF 5302/12E from Hong Kong RGC.

References

1. 2010 Census TIGER/Line Shapefiles. `http://www.census.gov/geo/www/tiger/tgrshp2010/tgrshp2010.html`.
2. L. Aalto, N. Göthlin, J. Korhonen, and T. Ojala. Bluetooth and WAP push based location-aware mobile advertising system. In *ACM MobiSys*, 2004.
3. G. AdWords. `http://adwords.google.com/`.
4. G. Aggarwal, G. Goel, C. Karande, and A. Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *SIAM SODA*, 2011.
5. N. Armenatzoglou, S. Papadopoulos, and D. Papadias. A general framework for geo-social query processing. In *VLDB*, 2013.
6. A. S. Asratian, T. M. J. Denley, and R. Häggkvist. *Bipartite Graphs and Their Applications.* Cambridge University Press, 2008.
7. R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press/Addison-Wesley, 1999.
8. J. Bao and M. F. Mokbel. GeoRank: An efficient location-aware news feed ranking system. In *ACM SIGSPATIAL GIS*, 2013.
9. J. Bao, M. F. Mokbel, and C.-Y. Chow. GeoFeed: A location-aware news feed system. In *IEEE ICDE*, 2012.
10. N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: A multidimensional workload-aware histogram. In *ACM SIGMOD*, 2001.
11. Y. Cai and T. Xu. Design, analysis, and implementation of a large-scale real-time location-based information sharing system. In *ACM MobiSys*, 2008.
12. X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. In *VLDB*, 2010.
13. B. Chandramouli and J. Yang. End-to-end support for joins in large-scale publish/subscribe systems. In *VLDB*, 2008.
14. B. Chandramouli, J. Yang, P. K. Agarwal, A. Yu, and Y. Zheng. ProSem: Scalable wide-area publish/subscribe. In *ACM SIGMOD*, 2008.
15. J.-J. Chen, J. Wu, C.-S. Shih, and T.-W. Kuo. Approximation algorithms for scheduling multiple feasible interval jobs. In *IEEE RTCSA*, 2005.
16. F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, J. Sgall, and T. Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *Journal of Discrete Algorithms*, 4(2):255–276, 2006.

17. C.-Y. Chow, J. Bao, and M. F. Mokbel. Towards location-based social networking services. In *ACM SIGSPATIAL LBSN*, 2010.
18. C.-Y. Chow, M. F. Mokbel, J. Nap, and S. Nath. Evaluation of range nearest-neighbor queries with quality guarantee. In *SSTD*, 2009.
19. G. Cong, C. S. Jensen, and D. Wu. Efficient Retrieval of the Top-k Most Relevant Spatial Web Objects. In *VLDB*, 2009.
20. R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 43(4), 2011.
21. I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *IEEE ICDE*, 2008.
22. G. M. Djuknic and R. E. Richton. Geolocation and Assisted GPS. *IEEE Computer*, 34(2):123–125, 2001.
23. Facebook. `http://www.facebook.com/about/location`.
24. J. Feldman, N. Korula, V. Mirrokni, S. Muthukrishnan, and M. Pál. Online ad assignment with free disposal. In *Internet and Network Economics*, 2009.
25. Foursquare. `http://www.foursquare.com`.
26. D. Freni, C. R. Vicente, S. Mascetti, C. Bettini, and C. S. Jensen. Preserving location and absence privacy in geo-social networks. In *ACM CIKM*, 2010.
27. Google Buzz Mobile. `http://www.google.com/mobile/buzz`.
28. G. Grimmett. *Probability: an introduction*. Oxford University Press, 1986.
29. P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE TSSC*, 4(2):100–107, 1968.
30. H. Jeung, M. L. Yiu, X. Zhou, and C. S. Jensen. Path prediction and predictive range querying in road network databases. *VLDB Journal*, 19(4):585–602, 2010.
31. M.-Y. Kao, T.-W. Lam, W.-K. Sung, and H.-F. Ting. A decomposition theorem for maximumweight bipartite matchings with applications to evolutionary trees. In *Algorithms-ESA*, 1999.
32. M.-Y. Kao, T.-W. Lam, W.-K. Sung, and H.-F. Ting. A decomposition theorem for maximum weight bipartite matchings. *SIAM Journal on Computing*, 31(1):18–26, 2001.
33. R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *ACM STOC*, 1990.
34. T. Kesselheim, K. Radke, A. Tönnis, and B. Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In *Algorithms–ESA 2013*, 2013.
35. A. Khoshgozaran and C. Shahabi. Private buddy search: Enabling private spatial queries in social networks. In *IEEE SIN*, 2009.
36. H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
37. K. W.-T. Leung, D. L. Lee, and W.-C. Lee. CLR: A collaborative location recommendation framework based on co-clustering. In *ACM SIGIR*, 2011.
38. J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel. LARS: A location-aware recommender system. In *IEEE ICDE*, 2012.
39. Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang. IR-Tree: An efficient index for geographic document search. *IEEE Trans. on Knowledge and Data Engineering*, 23:585–599, 2011.
40. Loopt. `http://www.loopt.com`.
41. M. Mahdian and Q. Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *ACM STOC*, 2011.
42. C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
43. A. Mehta. Online matching and ad allocation. *Theoretical Computer Science*, 8(4):265–368, 2012.
44. A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani. Adwords and generalized on-line matching. In *IEEE FOCS*, 2005.
45. A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani. Adwords and generalized online matching. *Journal of the ACM*, 54(5), 2007.
46. M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In *ACM SIGMOD*, 2004.
47. A. Noulas, S. Scellato, C. Mascolo, and M. Pontil. Exploiting semantic annotations for clustering geographic areas and users in location-based social networks. In *International AAAI Conference on Weblogs and Social Media*, 2011.

48. O. Phelan, K. McCarthy, and B. Smyth. Using twitter to recommend real-time topical news. In *ACM RecSys*, 2009.
49. Renren. `http://www.renren.com`.
50. L. Siksnys, J. Thomsen, S. Saltenis, and M. L. Yiu. Private and flexible proximity detection in mobile social networks. In *IEEE MDM*, 2010.
51. A. Silberstein, J. Terrace, B. F. Cooper, and R. Ramakrishnan. Feeding Frenzy: Selectively materializing user's event feed. In *ACM SIGMOD*, 2010.
52. Sina Weibo. `http://www.weibo.com`.
53. J.-W. Son, A. Kim, and S.-B. Park. A location-based news article recommendation with explicit localized semantic analysis. In *ACM SIGIR*, 2013.
54. Twinkle. `http://tapulous.com/twinkle`.
55. D. B. West. *Introduction to Graph Theory (2nd Edition)*. Prentice Hall, 2001.
56. D. Wu, G. Cong, and C. S. Jensen. A framework for efficient spatial web object retrieval. *VLDB Journal*, 21:797–822, 2012.
57. D. Wu, M. L. Yiu, and C. S. Jensen. Moving spatial keyword queries: Formulation, methods, and analysis. *ACM Trans. on Database Systems*, 38:7:1–7:47, 2013.
58. M. Ye, P. Yin, and W.-C. Lee. Location recommendation for location-based social networks. In *ACM SIGSPATIAL GIS*, 2010.
59. M. Ye, P. Yin, W.-C. Lee, and D.-L. Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. In *ACM SIGIR*, 2011.
60. H. Yin, Y. Sun, B. Cui, Z. Hu, and L. Chen. Lcars: a location-content-aware recommender system. In *Proc. of the ACM Conference on Knowledge Discovery and Data Mining*, 2013.
61. D. Zhang, Y. M. Chee, A. Mondal, A. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *IEEE ICDE*, 2009.
62. V. W. Zheng, Y. Zheng, X. Xie, and Q. Yang. Collaborative location and activity recommendations with GPS history data. In *WWW*, 2010.
63. Y. Zheng, X. Xie, and W.-Y. Ma. GeoLife: A collaborative social networking service among user, location and trajectory. *IEEE Data Engineering Bulletin*, 33(2):32–39, 2010.
64. Y. Zheng, L. Zhang, Z. Ma, X. Xie, and W.-Y. Ma. Recommending friends and locations based on individual location history. *ACM Trans. on the Web*, 5(1):5, 2011.
65. Y. Zhou, A. Salehi, and K. Aberer. Scalable delivery of stream query results. In *VLDB*, 2009.