Efficient Retrieval of the Top-k Most Relevant Event-Partner Pairs

Wu, Dingming; Xiao, Erjia; Zhu, Yi; Jensen, Christian S.; Lu, Kezhong

Published in:
IEEE Transactions on Knowledge and Data Engineering

DOI (link to publication from Publisher):
10.1109/TKDE.2021.3118552

Publication date:
2023

Document Version
Accepted author manuscript, peer reviewed version

Efficient Retrieval of the Top-k Most Relevant Event-Partner Pairs

Dingming Wu, Erjia Xiao, Yi Zhu, Christian S. Jensen, and Kezhong Lu✉

Abstract—The proliferation of event-based social networking (EBSN) motivates studies on topics such as event, venue, and friend recommendation as well as event creation and organization. In this setting, the notion of event-partner recommendation has attracted attention. When recommending an event to a user, this functionality allows the recommendation of partners with whom to attend the event. However, in existing proposals, recommendations are pushed to users at the system's initiative. In contrast, EBSNs provide users with keyword-based search functionality. This way, users may retrieve information in pull mode. We propose a new way of accessing information in EBSNs that combines pull and push, thus allowing users to not only conduct ad-hoc searches for events, but also to receive partner recommendations for retrieved events. Specifically, we define and study top-k event-partner (kEP) pair retrieval querying that integrates keyword-based search for events with event-partner recommendation. This type of query retrieves event-partner pairs, taking into account the relevance of events to user-supplied keywords and so-called together preferences that indicate the extent of a user's preference to attend an event with a given partner. To compute kEP queries efficiently, we propose a rank-join based framework with three optimizations. Results of empirical studies with implementations of the proposed techniques demonstrate that the proposed techniques are capable of excellent performance.

Index Terms—Social networking, query processing.

1 Introduction

The recent proliferation of event-based social networking (EBSN), as exemplified by Meetup[1] and Eventbrite[2], has not gone unnoticed in the research community, where substantial efforts have been devoted to the recommendation of events [1], [2], [3], [4], [5], venues [6], [7], [8], [9], [10], and friends [11], [12]. Unlike previous recommendation techniques that each focus on recommending only one type of items (either events or friends), event-partner recommendation [13], [14] aims to recommend events together with partners. Attending an event with a partner may be more attractive than attending an event alone, and a user may not attend an event if the user has to do so alone. However, recommendations are delivered in push mode based on historical data. This means that when users are interested in new events, not reflected in their historical data, the push-based solutions face the well-known cold-start problem. Although studies exist that address the cold-start problem [15], [16], push-based solutions fail to provide interesting event-partner recommendations in response to ad-hoc user needs.

This paper addresses the cold-start problem by integrating ad-hoc search and recommendation. This yields a new way of accessing information in EBSNs that allows users to not only perform ad-hoc event searches, but also to receive partner recommendations for retrieved events. Given user-specified keywords and historical data related to a user, this functionality, called the top-k event-partner (kEP) pair retrieval query, retrieves k event-partner pairs, consisting of events that are relevant to the keywords and partners with whom the user is willing to attend the events. For instance, a user "Jordan" who wants to know about events related to "rock concert" can search for events matching the keywords "rock concert," and then a partner is recommended for each retrieved event. This query differs from keyword queries that retrieve relevant events without partners, and it differs from event-partner recommendation that uses only historical data.

The query retrieves k event-partner pairs that score the highest according to a scoring function with two components. One concerns the text relevance of events w.r.t. the query keywords; here, any existing models, e.g., language models [17], can be used. The other takes into account so-called together preferences that capture whether the user and a potential partner are willing to attend an event together. In one study [13], the together preference of users u_q and u for event e takes into account the number of events that are similar to e and that u_q and u have both attended. This definition disregards the similarity between users, and if users search for events that are not relevant to their historical data, there will be no valid event-partner pairs. Thus, to facilitate the ad-hoc search setting, we propose a new approach to determining together preferences that takes the similarity between the historical attendances of users into account. The paper's empirical study provides a detailed comparison between the new approach and the existing approach.

The kEP query can be seen as a top-k join query that returns the k highest scoring pairs of events and users that

- *D. Wu, K. Lu, E. Xiao, and Y. Zhu are with the College of Computer Science and Software Engineering, Shenzhen University, China. E-mail: {dingming, kzlu}@szu.edu.cn, erjiaxiao@gmail.com, zhuyi2016@email.szu.edu.cn*
- *C. S. Jensen is with the Department of Computer Science, Aalborg University, Denmark. E-mail: csj@cs.aau.dk*

1. http://www.meetup.com
2. http://www.eventbrite.com

join. The kEP query can then be computed by first joining events and users and then, for each event e, choose the k highest scoring pairs of e and a user as result candidates. Next, the candidate pairs are ranked across events, and the k pairs with the highest score are returned. However, this method is inefficient because the join considers all combinations of events and users. Another approach to computing kEP queries is to extend the rank-join algorithm [18]. Here, the idea is to scan input events and users ordered according to scoring functions. However, while events may be ordered according to the relevance of their textual descriptions to the query keywords, there is no obvious scoring function for the partners with which to attend events.

We propose a rank-join based framework for computing kEP queries where the scoring predicate for the partners (users) is the number of events that they have attended. The intuition is that users who have attended many events tend to have high together preferences (cf. Section 4). We study two representative join strategies, nested loop join and ripple join [19], within the framework. To improve the performance of the framework further, we propose three optimizations. First, an unpromising-event pruning technique enables disregarding events that cannot enter the result. Next, to reduce the computational cost of finding the partner with the highest together preference for an event, we propose two algorithms: one that uses an Event-Member List (EML) data structure and one that uses a Shared-Event User Graph (SEUG) data structure. We also combine these optimizations and propose a more efficient partner computation algorithm that also includes an effective event pruning technique. To obtain insight into pertinent properties of the framework and its optimizations, we conduct experiments with an implementation of the framework using real data. The study offers evidence that the framework is useful in practice.

The paper extends our previous work [20] by proposing a new way of measuring the together preference (Equation 2) that takes the similarity between the historical attendances of users into account, while our previous work adopts an existing method [13] to compute together preferences. The experimental results provide evidence that the proposed method outperforms the existing method in terms of recommendation accuracy. Since the kEP query in this paper adopts a different together preference from the previous work, the key partner optimization in the previous work is no long applicable and has been discarded. Instead, this paper proposes a new optimization technique based on the SEUG data structure for efficient partner computation and also provide an efficient algorithm that combines the use of the SEUG data structure and the EML data structure for partner computation (Section 5.3).

The rest of this paper is organized as follows. Section 3 formally defines the top-k event-partner retrieval problem. Section 4 presents the solution framework. The three optimizations are detailed in Section 5. We report on a performance evaluation in Section 6. Finally, we cover related work in Section 2 and offer conclusions and research directions in Section 7.

2 Related Work

2.1 Recommendations on Social Platform

A large number of social-platform recommendation techniques exist. They differs in their objectives and the methods employed. Based on the objectives, the studies relevant to our work can be categorized roughly into event recommendation, friend recommendation, group recommendation, and event-partner recommendation. We review each category of related work in turn and characterize the relations to this paper at the end.

Event Recommendation. Event recommendation suggests the most relevant events for users to participate in. For this purpose, various methods have been proposed. Qiao et al. [4] propose a Bayesian probability model that takes social impact and implicit feedback into account. Qiao et al. [5] present a Bayesian latent factor model that unifies online and offline social relationships, geographical feature of events, and implicit rating data from users. Next, SogBmf [1] is a Bayesian latent factor model that combines social group influence and individual preferences for event recommendation. Further, HeteRS [21] is a general graph-based model that can recommend groups to users, recommend tags to groups, and recommend events to users in one framework. LCARS [22] offers a particular user a set of venues or events, giving consideration to both personal interests and local preferences. Macedo et al. [23] exploit several contextual signals for learning to rank events for personalized recommendation, including content-based signals from event' descriptions, collaborative signals from user' relationships, social signals from group memberships, and location and temporal signals. CBPF [2] is a collective Bayesian Poisson factorization model for handling the problem of cold-start event recommendation.

Friend Recommendation. Friend recommendation suggests user-user relationships, which may involve estimation of the likelihood that two non-friends will become friends. Wan et al. [11] recommend friends according to informational utility, which represents the degree to which a friend satisfies the target user's unfulfilled informational needs. Lu et al. [12] propose a Bayesian latent factor model, which jointly formulates geographical information, implicate user ratings, and user behavior, for friend recommendation. Yu et al. [24] propose an end-to-end social recommendation framework based on Generative Adversarial Nets (GAN), which can dynamically identify reliable social relations under the supervision of the seeded friends.

Group Recommendation. Group recommendation explores the preferences of a group of users in relation to individual items. Gorla et al. [25] combine the relevances of items to individual group members with the relevance of items to the group as a whole. The COnsensus Model [26] models the generative process of group activities, in which each group has a multinomial distribution over latent topics, and these topics attract a set of users to join. Group recommendations are based on COM, which is able to exploit both users' selection history and users' personal considerations of content factors. SIGR [27] adopt an attention mechanism to learn each user's social influence and integrate users' global and local social network structure information to estimate a user's social influence. Guo et al. [28] propose a social self-

attention based aggregation strategy that models directly the interactions among group members. Du et al. [29] formalize group recommendation as a ranking problem and propose a group event recommendation framework GERF based on the learning-to-rank technique. CVTM [30] captures group interests in an event from an event's content and venue. The correlation between organizer and content is used to alleviate the sparsity of textual content.

Event-Partner Recommendation. Event-partner recommendation [13], [14] helps users find interesting events and suitable partners for attending events. Tu et al. [13] provide partner recommendations based on users' historical attendance preferences, social contexts, and geographic information. Yin et al. [14] propose a generic graph-based embedding model to collectively embed relations among users, events, locations, time, and text content into a shared, low-dimension space.

Relation to Recommendation Studies. Most of the above studies concern the recommendation of only one type of item, often events or friends, while our study concerns the recommendation of events and event partners. Thus, the two previous studies on event-partner recommendation are most relevant to our study. Our proposal differs from these in several respects. Tu et al. [13] recommend a partner for a given target user and event. Yin et al. [14] suggest event-partner pairs for a given target user based on information from an EBSN. They propose purely recommendation-based methods, while we combine search and recommendation. The kEP query in our study takes keywords provided by a target user as parameters and returns event-partner pairs such that the events are relevant to the query keywords and the target user is likely to be willing to participate in the event with the partner.

This paper extends our previous work [20]. The new contributions include a new way of computing together preferences; efficient partner computation based on SEUG, EML+SEUG, and unpromising-Event Pruning+SEUG; and an empirical study that compares the accuracy of the new together preference function and the original one and evaluates algorithm efficiency.

2.2 Rank-Join Algorithms

Based on the A* optimization strategy, the J^* algorithm [31] enables querying of ordered data sets by means of user-defined join predicates. NRA-RJ [32] is a pipelined query operator that produces a global rank from ranked input streams based on a scoring function. Ilyas et al. [18] rank join results progressively during join operations, exploiting the individual orders of the inputs.

Ranking (top-k) queries have also been integrated into relational database systems [33], [34]. Mamoulis et al. [35] identify two phases that any (non-random access) NRA algorithm should go through: a growing phase and a shrinking phase. Their LARA algorithm employs a lattice to minimize the computational cost during the shrinking phase. The FRPA rank join operator [36] allows efficient computation of score bounds on unseen join results and prioritizes the I/O requests of the rank join operator based on the potential of each input to generate results with high scores. The Pull/Bound Rank Join (PBRJ) [37] is an algorithm template that generalizes previous rank join algorithms. The idea is to alternate between pulling tuples from input relations and upper bounding the score of join results that use the unread part of the input. The join results collected as tuples are pulled, and the algorithm stops once the top-k buffered results have a score at least equal to the upper bound.

We extend the state-of-the-art rank-join algorithm [18] and propose a framework with optimizations that supports the efficiently processing of kEP queries.

2.3 Query Processing on Social Networks

Recently, various methods have been proposed for querying social networks. TCS-SSN [38] retrieves communities from spatio-social networks that cover given keywords and that are characterized by high social influence and small traveling time. Co-located community search [39] considers users' spatial information in k-truss search that aims to find highly correlated user groups in social networks. Multi-attribute community (MAC) search [40] finds communities that are highly relevant to query users and have top overall scores for multiple attributes according to user preferences. Kou et al. [41] address team formation in social network by taking into account both structure constraints and communication constraints related to team members. Li et al. [42] retrieve skyline cohesive groups in social networks, in which each group cannot be dominated by any other group in terms of social and spatial cohesiveness. Targeting attributed community search, the parameter-free contextual community model [43] uses a set of keywords describing a desired matching community context and retrieves the community that is both structure and attribute cohesive w.r.t. the provided query context. The CGNN query [44] incorporates cohesive social relationships into group nearest neighbor (GNN) search over road-social networks such that both the query users of highest closeness and the corresponding top-j objects are retrieved.

The querying methods underlying the above queries are generally designed specifically for the proposed queries, and the methods differ from the methods we propose. As a result, these methods cannot be applied to our problem.

3 Problem Definition

An event-based social network (EBSN) can be modeled as a bipartite graph $G = (U, E, R)$, where U represents a set of users, E represents a set of events posted by the users, and $R \subseteq U \cdot E$ is set of participation relationships between users and events, i.e., $r = (u, e) \in R, u \in U, e \in E$. Each event $e \in E$ is associated with a text document $e.\psi$ that describes the content and features of the event. Specifically, we assume that a document is represented by a term vector [45]. The members of an event e are the users u who have joined $e : \{u|(u, e) \in R\}$.

A **top-$\underline{k}$ most relevant $\underline{\text{E}}$vent-$\underline{\text{P}}$artner pair retrieval (kEP)** query $Q = (k, u_q, \psi_q)$ takes three parameters: (i) k is the number of requested event-partner pairs, (ii) u_q is a query user, and (iii) ψ_q is a set of query keywords. Let $t(\psi_q, e.\psi)$ be the textual relevance (e.g., defined using language models [17]) of event e w.r.t. the query keywords ψ_q. The result

of a kEP query contains k event-partner pairs (e, u) with the highest score $f(u_q, \psi_q, e, u)$ (Equation 1). The events in the result are distinct, but the same partner user may be paired with multiple events. The scoring function considers both textual relevance $t(\psi_q, e.\psi)$ and together preference $p(u^*, e, u)$. The together preference $p(u^*, e, u)$ measures the probability that user u^* is willing to attend event e with user u, $u^* \neq u$. Ties are broken arbitrarily.

We use a scoring function that takes the form of a weighted sum of the textual relevance and the together preference. The techniques we propose are, however, applicable to any scoring function that is monotone in terms of both the textual relevance and the together preference. The kEP query finds event-partner pairs (e, u) such that the events are relevant to the query keywords and the query user is likely to participate in the events with partners.

$$f(u_q, \psi_q, e, u) = \alpha \cdot t(\psi_q, e.\psi) + (1 - \alpha) \cdot p(u_q, e, u)$$
$$s.t.\ \ t(\psi_q, e) \in (0, 1] \ \ \wedge \ \ p(u_q, e, u) \in (0, 1] \quad (1)$$

The together preference function $p(u^*, e, u)$ is defined in Equation 2, and it is motivated by two observations. First, a user may wish to attend an event that is similar to events that the user has previously participated in. Second, people tend to join an event with a partner with whom they share common interests. In our scenario, the common interests are captured by the common events that two people have participated in.

$$p(u^*, e, u) = \frac{\sum_{e_i \in N(e)} s(e, e_i) \cdot b(e_i, u) \cdot J(E_{u^*}, E_u)}{\sum_{e_i \in N(e)} s(e, e_i)} \quad (2)$$

Function $p(u^*, e, u)$ takes three arguments, i.e., a target user u^*, an event e, and a partner user u. The range of $p(u^*, e, u)$ is $[0, 1]$. Large values of $p(u^*, e, u)$ indicate that target user u^* is very likely to want to attend event e with partner user u. In the definition, $N(e)$ is the neighborhood of an event e, which is the set of events e_i whose document similarity $s(e, e_i)$ is no less than a threshold τ. We define the similarity $s(e, e_i)$ as cosine similarity [46]. However, the proposed method is independent of the choice of the similarity measure, and any reasonable similarity function can be adopted. Given a partner user u and an event e_i, $b(e_i, u) = 1$ if u has participated in event e_i; otherwise, $b(e_i, u) = 0$. The common interests between u^* and u is defined as the Jaccard similarity coefficient $J(E_{u^*}, E_u)$ [47] of the sets of events E_{u^*} and E_u that u^* and u has attended, respectively. The denominator is the sum of the similarities between event e and each event e_i in the neighborhood $N(e)$. The together preference function is not symmetric, i.e., $p(u^*, e, u) \neq p(u, e, u^*)$. The together preference function can be interpreted in three steps. First, an event e is taken as a candidate event for user u^*. Second, a user u is considered a candidate partner for u^* if the two users have common interests, i.e., $J(E_{u^*}, E_u) > 0$. Third, u should have participated in events similar to e, i.e., $b(e_i, u) = 1$, where e_i is in $N(e)$.

In an EBSN, some events may occur periodically, e.g., weekly or monthly. A user may attend same such event multiple times. Hence, in the result of a kEP query, events may exist that have been attended previously by the query user. We do not exclude such events, since the query user may be interested in them and may attend them again.

Example 1. *Consider the EBSN in Figure 1, where there are five users $U = \{u_1, u_2, \cdots, u_5\}$ and five events $E = \{e_1, e_2, \cdots, e_5\}$. The participation relationship R between users and events is given by the edges. Table 1 shows the term vectors of the documents of the events, and Table 2 shows the similarities between the documents of the events. Given $\tau = 0.3$, the neighborhood of event e_3, $N(e_3)$, is $\{e_2, e_4, e_5\}$ because these events have similarity no less than 0.3 to e_3, as shown in Table 4. The together preference $p(u_4, e_3, u_3) = (0.5 + 0.6) \cdot 0.8/(0.5 + 0.3 + 0.6) = 0.63$ because u_3 has attended events e_2 and e_5 in neighborhood $N(e_3)$ and the Jaccard similarity coefficient of the sets of events E_{u_3} and E_{u_4} is 0.8–see Table 4.*

Example 2. *Continuing Example 1, given a kEP query Q with $k = 2$, $u_q = u_4$, and $\psi_q = \{t_1, t_3\}$, the textual relevance of each event w.r.t. the query keywords is shown in Table 3, the Jaccard similarity coefficient of the sets of events that are attended by users u_i and u_j are shown in Table 5, and the neighborhoods of events are shown in Table 4 (given $\tau = 0.3$). Given query user u_4, for each event e, we choose user u^e as a partner if the together preference $p(u_4, e, u^e)$ exceeds the together preference of any other user. Table 4 shows the partner for each event and the corresponding together preference. Given $\alpha = 0.5$, the top-2 event-user pairs of query Q are (e_2, u_3) and (e_3, u_3) with scores 0.75 and 0.71, respectively.*

4 Rank-Join based Framework

We proceed to present the rank-join based framework for the processing of kEP queries. Section 4.1 presents the main data structures used in the framework. Section 4.2 explains the query processing algorithm, and Section 4.3 covers the two join strategies in the framework.

4.1 Data Structures

The rank-join based framework includes two main data structures. One is a representation of the event-user graph G that is stored in main memory. The other is a disk-resident inverted index II_d that indexes the documents of all events in the EBSN. The inverted index consists of two main components: (1) a vocabulary of all distinct terms in the collection of documents and (2) a posting list for each term t in the vocabulary. Each posting list is a sequence of pairs (id, w), where id identifies an event e whose document $e.\psi$ contains term t and w is the weight of term t in document $e.\psi$.

4.2 Query Processing Algorithm

Following the idea of the rank-join algorithm [18], the framework conducts a join operation on the ranked input events and users. The ranked input events are obtained by issuing a keyword query using the inverted index II_d. The relevant events are retrieved in descending order of their textual relevance.

In contrast, there is no straightforward way to obtain the ranked input users. Following the definition of the together preference function (Equation 2), the framework

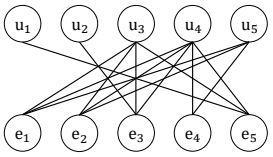


Fig. 1: Example EBSN

TABLE 1: Term Vectors

Event	Term Vector
e_1	t_2 t_3 t_4 t_9
e_2	t_1 t_3 t_7 t_8
e_3	t_1 t_3 t_5
e_4	t_2 t_3 t_6 t_9
e_5	t_1 t_3 t_4 t_5 t_6

TABLE 2: Similarities

$s(e_i, e_j)$	e_1	e_2	e_3	e_4	e_5
e_1	1	0.2	0.2	0.7	0.3
e_2	0.2	1	0.5	0.2	0.3
e_3	0.2	0.5	1	0.3	0.6
e_4	0.7	0.2	0.3	1	0.3
e_5	0.3	0.3	0.6	0.3	1

TABLE 3: Textual Relevances

Event	$t(\psi_q, e.\psi)$
e_1	0.4
e_2	0.7
e_3	0.8
e_4	0.3
e_5	0.6

TABLE 4: Neighborhoods and Partners

Event	$N(e_i)$	u^e	$p(u_4, e, u^e)$
e_1	e_4, e_5	u_5	0.42
e_2	e_3, e_5	u_3	0.8
e_3	e_2, e_4, e_5	u_3	0.63
e_4	e_1, e_3, e_5	u_3	0.8
e_5	e_1, e_2, e_3, e_4	u_3	0.64

TABLE 5: Jaccard Similarity Coefficients

$J(E_{u_i}, E_{u_j})$	E_{u_1}	E_{u_2}	E_{u_3}	E_{u_4}	E_{u_5}
E_{u_1}	1	0	0.25	0.2	0
E_{u_2}	0	1	0.25	0.2	0
E_{u_3}	0.25	0.25	1	0.8	0.4
E_{u_4}	0.2	0.2	0.8	1	0.6
E_{u_5}	0	0	0.4	0.6	1

uses a heuristic user scoring predicate, namely the number of events users have attended. The motivation is that users who have attended many events are expected to have high together preferences w.r.t. query user u_q and event e. This heuristic assumes that the attended events are similar to event e. Therefore, the framework retrieves users based on two constraints, i.e., (1) the retrieved users should have attended at least one event in neighborhood $N(e)$, and (2) the users are retrieved in descending order of the number of events they have attended. Specifically, the set of users $U(u_q, e)$ that is fed to the join is constructed as follows. For each retrieved event e, its neighborhood $N(e)$ is computed. Then, for each event in the neighborhood, its participants are obtained from the event-user graph. Among the members of all the events in neighborhood $N(e)$, a user is added to $U(u_q, e)$ if the user has attended at least one event in E_{u_q}. Next, the users in $U(u_q, e)$ are sorted descendingly on the number of events they have attended.

The query processing algorithm in the framework borrows the idea of the TA (Threshold Algorithm) [48] and consumes the input events and users to generate candidate event-user pairs. A threshold T is maintained that is calculated by setting the textual relevance to that of the upcoming event and the together preference to 1 in the scoring function (Equation 1). When the score of the k^{th} largest candidate pair is no less than T, the algorithm reports the obtained top-k pairs. It is straightforward to prove that threshold T serves as an upper bound on the scores of the event-partner pairs that have not yet been considered, since the events are retrieved in descending order of the textual relevance and because the maximum value of the together preference is set to 1. Algorithm 1 is the pseudo code of the query processing algorithm. Different join strategies can be adopted for the join in the algorithm.

Algorithm 1 kEP(u_q, ψ_q, k)

```
PriorityQueue Queue ← ∅
E_q ← GetAttendedEvent(u_q, G)
Threshold T ← ∞
while e ← GetNextEvent(ψ_q, II_d) do
    T ← α · t(ψ_q, e) + (1 − α) · 1
    if T ≤ the score of the k^th largest pair in Queue then
        break
    end if
    N(e) ← ∅
    for each event e_i ∈ E do
        if s(e, e_i) ≥ τ then
            Add e_i to N(e)
        end if
    end for
    U(u_q, e) ← {∪{U(a_i)|a_i ∈ N(e)}} ⋂ {∪{U(b_i)|b_i ∈ E_{u_q}}} \ {u_q}
    if U(u_q, e) is empty then
        continue
    end if
    Sort the users in U(u_q, e) in the descending order of the number of events they have attended.
    Candidate pairs P ← Join(e, U(u_q, e))
    Calculate the score f(u_q, ψ_q, e, u) of each candidate pair in P.
    Add all candidate pairs in P to Queue
end while
return the k pairs with the highest f() in Queue
```

4.3 Join Strategies

We consider two state-of-the-art join strategies in the framework, namely nested loop join and ripple join [19]. The nested loop join consists of a nested loop, where the outer loop consumes events in descending order of the textual relevance and the inner loop, executed for each (outer) event, consumes the users in $U(u_q, e)$. When the inner loop for an event finishes, the partner for the event with the highest together preference has been identified, and this event-partner pair is output as a candidate.

In the ripple join, e.g., the "square" version, a previously unseen tuple (user or event) is retrieved from each of the two input lists in each step; these new tuples are joined with all previously seen tuples and with each other. As in the nested loop join, events are retrieved in descending order of the textual relevance. Unlike in the nested loop join, the users to be retrieved are organized in a priority queue sorted descendingly on the number of events they have attended. For each upcoming event e, the priority queue is updated dynamically by adding its user set $U(u_q, e)$. The square version consumes one event and one user at a time. In the empirical study, we also evaluate the performance of the

"rectangular" version that consumes different numbers of events and users at a time.

5 OPTIMIZATIONS

Although the rank-join based framework take advantage of both the rank-join and the TA algorithm, it is still inefficient in some cases. For instances, it may take a long time to find the partner for an event if the number of users considered in the join process is large. In addition, before returning the top-k pairs, unnecessarily many candidate pairs may have been produced, which incurs high computational cost. We develop three performance optimizations.

5.1 Unpromising-Event Pruning

This optimization reduces the computational cost by pruning events early that can not contribute to top-k pairs. We derive a worst allowed together preference $p_w(e)$ (Definition 1) for a newly retrieved event, which is a necessary condition of the event being able to contribute to the top-k result. This value is a lower bound on the together preference of the events in the final result. We also derive a best possible together preference $p_{ub}(e)$ (Definition 2) for a newly retrieved event, which estimates the highest possible together preference of the event with its partner.

Definition 1. ***Worst Allowed Together Preference*** $p_w(e)$*: Given a query user* u_q *and query keywords* ψ_q*, let* f_k *be the score of the current* k^{th} *candidate event-partner pair. The worst allowed together preference* $p_w(e)$ *of* e *is defined as* $p_w(e) = (f_k - \alpha \cdot t(\psi_q, e))/(1-\alpha)$.

Lemma 1. *If* $\forall u \in (U \setminus \{u_q\})(p(u_q, e, u) \leq p_w(e))$*, event* e *cannot belong to the result [20].*

Proof. Given event e, if $\forall u \in (U \setminus \{u_q\})(p(u_q, e, u) \leq p_w(e))$, we can derive that if $p(u_q, e, u) \leq (f_k - \alpha \cdot t(\psi_q, e))/(1-\alpha)$ then $\alpha \cdot t(\psi_q, e) + (1-\alpha) \cdot p(u_q, e, u) \leq f_k$, which means that event e cannot find a partner with a better score than f_k. Since f_k is the score of the current k^{th} candidate event-partner pair, event e cannot belong to the result. □

Definition 2. ***Best Possible Together Preference*** $p_{ub}(e)$*: Let* $N(e)$ *be the neighborhood of* e*, and define* $m = \max_{u \in (U \setminus \{u_q\})}\{|E_c| \mid E_c = N(e) \cap E_u\}$ *and* $r = \max_{u \in (U \setminus \{u_q\})}\{|E_c| \mid E_c = E_{u_q} \cap E_u\}$*, where* E_u *is the set of events that user* u *has attended. The best possible together preference* $p_{ub}(e)$ *of* e *is given as follows.*

$$p_{ub}(e) = \frac{\sum_{e_i \in TopM(u_q, e)} s(e, e_i)}{\sum_{e_i \in N(e)} s(e, e_i)} \cdot \frac{r}{|E_{u_q}|}, \tag{3}$$

where $|TopM(u_q, e)| = m$*,* $TopM(u_q, e) \subseteq N(e)$*, and* $\forall e_i \in TopM(u_q, e)\ \forall e_j \in (N(e) \setminus TopM(u_q, e))\ (s(e, e_i) \geq s(e, e_j))$.

Lemma 2. *The best possible together preference* $p_{ub}(e)$ *of* e *is an upper bound on the together preference of* e *with its partner.*

Proof. According to Equation 2, the numerator of the together preference sums up the similarities between the events e_i that user u has attended in neighborhood $N(e)$. This can be rewritten as follows: $\sum_{e_i \in N(e) \cap E_u} s(e, e_i)$. Set $TopM(u_q, e)$ contains the top-m similar events in neighborhood $N(e)$. Since $m = \max_{u \in (U \setminus \{u_q\})}\{|E_c| \mid E_c = N(e) \cap E_u\}$ is the maximum number of events attended by a user in $N(e)$ and $\forall u \in (U \setminus \{u_q\})(|TopM(u_q, e)| \geq |N(e) \cap E_u|)$, we can derive that $\forall u \in (U \setminus \{u_q\})(\sum_{e_i \in N(e) \cap E_u} s(e, e_i) \leq \sum_{e_i \in TopM(u_q, e)} s(e, e_i))$. Since $r = \max_{u \in (U \setminus \{u_q\})}\{|E_c| \mid E_c = E_{u_q} \cap E_u\}$ is the maximum number of events attended by both u_q and another user, we have $\forall u \in (U \setminus \{u_q\})(r/|E_{u_q}| \geq J(E_{u_q}, E_u))$. The denominators of $p_{ub}(e)$ and the together preference are the same. This proves that $\forall u \in (U \setminus \{u_q\})(p_{ub}(e) \geq p(u_q, e, u))$. □

Example 3. *Given query user* u_4*, we compute the best possible together preference of event* e_3*. Neighborhood* $N(e_3) = \{e_2, e_4, e_5\}$*,* $m = 2$*, and* $r = 4$*. Then we have* $TopM(u_4, e_3) = \{e_2, e_5\}$ *and* $r/|E_{u_4}| = 4/5 = 0.8$*. The best possible together preference is calculated as* $p_{ub}(e_3) = 0.8 \cdot (0.5 + 0.6)/(0.5 + 0.3 + 0.6) = 0.63$.

Lemmas 1 and 2 present the properties of $p_w(e)$ and $p_{ub}(e)$, respectively. Using these lemmas, Pruning Rule 1 is able to prune events that cannot contribute to pairs in the top-k result, thus enabling reduction of the computational cost.

Pruning Rule 1. *Given query user* u_q *and query keywords* ψ_q*, for event* e*, if* $p_w(e) > p_{ub}(e)$*, event* e *cannot contribute to a result pair and can be pruned.*

5.2 Partner Computation based on EML

The operation of finding a partner for an event is expensive. In particular, the cost is high when user set $U(u_q, e)$ considered in the join is large. The following optimization provides a way of finding the partner for an event without examining each user in set $U(u_q, e)$. The optimization uses an Event-Member List (EML) for each event that consists of pairs (u, num) sorted descendingly on num, the number of events attended by user u.

Table 6 shows the event-member lists of the five events in Figure 1. For instance, event e_4 has members u_4 and u_5 that have attended 5 and 3 events, respectively..

TABLE 6: Event-Member Lists

Event	Member List
e_1	$(u_4, 5), (u_3, 4), (u_5, 3)$
e_2	$(u_4, 5), (u_3, 4), (u_5, 3)$
e_3	$(u_4, 5), (u_3, 4), (u_2, 1)$
e_4	$(u_4, 5), (u_5, 3)$
e_5	$(u_4, 5), (u_3, 4), (u_1, 1)$

Algorithm 2 shows the pseudo code of the efficient partner computation. It takes a query user u_q, an event e, and a neighborhood $N(e)$ as arguments, and it returns the partner u who maximizes the together preference $p(u_q, e, u)$. Given neighborhood $N(e)$, the member lists of the events in the neighborhood are fetched (line 4). Function **GetNextPair**() chooses the pair $(u, num), u \neq u_q$ with the largest num from the first elements of all fetched member lists. If multiple pairs have the same largest num, the pair from the member list of event e_i with the largest similarity $s(e, e_i)$ is selected (line 6). The together preference

$p(u_q, e, u)$ is computed, and pair (u, num) is removed from each member list that contains it. If $p(u_q, e, u)$ exceeds the together preference p_1 of the current candidate partner u_p, user u is taken as the candidate partner (lines 7–10). Then function **GetNextPair**() is called again to obtain the next pair (u, num) that is used to compute an upper bound p_r (Lemma 3) on the together preferences of the rest of the users in the fetched member lists (lines 12–14). If the together preference p_1 of the current candidate partner u_p is no less than p_r, user u_p is the partner who maximizes $p(u_q, e, u)$ and is returned (Pruning Rule 2). Otherwise, the algorithm repeats the above process.

Algorithm 2 ComputePartner$(N(e), u_q, e)$

```
1: p1 ← −1
2: pr ← +∞
3: up ← null
4: Fetch the member list ml(ei) of each event in N(e)
5: while p1 < pr do                                  ▷ Pruning Rule 2
6:     (u, num) ← GetNextPair()
7:     if p1 < p(uq, e, u) then
8:         p1 ← p(uq, e, u)
9:         up ← u
10:    end if
11:    Remove (u, num) from the member list
12:    (u, num) ← GetNextPair()
13:    x ← min{num, |N(uq, e)|}
14:    Compute pr according to Lemma 3
15: end while
16: return up
```

Lemma 3. *Given a query user u_q and an event e, let (u, num) be the pair returned by function* **GetNextPair**() *and define $x = \min\{num, |N(e)|\}$. Then set $TopX(u_q, e)$ contains the top-x events $\{e_i\}$ in neighborhood $N(e)$ with the largest similarity $s(e, e_i)$. An upper bound on the together preference of the users in the member lists of the events in $N(e)$ is*

$$p_r = \frac{\sum_{e_i \in TopX(u_q,e)} s(e, e_i)}{\sum_{e_i \in N(e)} s(e, e_i)} \cdot \frac{r}{|E_{u_q}|}, \tag{4}$$

where r is defined in Definition 2.

Proof. Recall that (i) the pairs (u, num) in the member list are sorted descendingly on num and that (ii) function **GetNextPair**() returns the pair with the largest num from the first elements in all member lists of the events in $N(e)$. This means that no user in the member lists of the events in $N(e)$ can have attended more events than the returned num. Since $x = \min\{num, |N(e)|\}$, no user in the member lists of the events in $N(e)$ has attended more than x events in $N(e)$. Given that set $TopX(u_q, e)$ contains the top-x events $\{e_i\}$ in neighborhood $N(e)$ with the largest similarity $s(e, e_i)$ and $\forall u \in (U \setminus \{u_q\})(r/|E_{u_q}| \geq J(E_{u_q}, E_u))$ which has been proved in Lemma 2, then, for any user u_i in the member lists of the events in $N(e)$, we have $p_r \geq p(u_q, e, u_i)$. □

Pruning Rule 2. *In Algorithm 2, let p_1 be the together preference of the current candidate partner u_p. If $p_1 \geq p_r$, user u_p is returned as the partner of event e, and no other user in the fetched member lists can be the partner and can be pruned.*

Example 4. *Given query user u_4 and event e_2, according to Table 4, neighborhood $N(u_4, e_2) = \{e_3, e_5\}$. According to Table 6, the pair returned by function* **GetNextPair**() *is $(u_3, 4)$. The together preference $p(u_4, e_2, u_3) = 0.8 \cdot (0.5+0.6)/(0.5+0.3+0.6) = 0.63$. Pair $(u_3, 4)$ is removed from the member list of each event in $N(u_4, e_2)$. User u_3 is taken as the candidate partner. Next,* **GetNextPair**() *returns $(u_2, 1)$. We have $x = 1$ and $p_r = (0.5/(0.5 + 0.3)) \cdot 0.2 = 0.13$. Since $p(u_4, e_2, u_3) \geq p_r$, no other user can have higher together preference than does user u_3. Finally, user u_3 is returned as the partner for u_4 at e_2.*

5.3 Partner Computation based on SEUG

According to the definition of the kEP query, the partner of an event is the user who maximizes the together preference (Equation 2). A candidate partner may have a high together preference if (i) the number of events in $N(e)$ that both the query user and the candidate partner attended is large and if (ii) the query user and the candidate partner share many historical events. The partner computation based on the EML proposed in Section 5.2 uses the number of events the users have attended as the scoring predicate, which relates only item (i) above. The algorithm for partner computation presented here uses a Shared-Event User Graph (SEUG) that related to item (ii) above. We will eventually provide a method that combines EML and SEUG, resulting in a more efficient algorithm.

An SEUG G^u is an undirected, weighted graph. The vertices in G^u are the users in the EBSN. An edge (u_i, u_j) is present if users u_i and u_j have co-attended at least one event, i.e., $E_{u_i} \cap E_{u_j} \neq \emptyset$. The weight of edge (u_i, u_j) is the number of events u_i and u_j have co-attended. Graph G^u is stored as adjacency lists. For each user, the edges in its adjacency list are ordered descendingly on their weights. The adjacency list of a user is called the user's partner list. Given a user u, only the users in their partner lists can be the partner of u for any event. This is because the users who are not in the partner list of u have not co-attended any event with u, so the together preferences of these users and u for any event is 0.

Example 5. *Figure 2 shows the SEUG G^u for the five users in Figure 1. Table 7 shows the adjacency lists of graph G^u. The users in the partner lists are ordered descendingly on the number of shared events.*

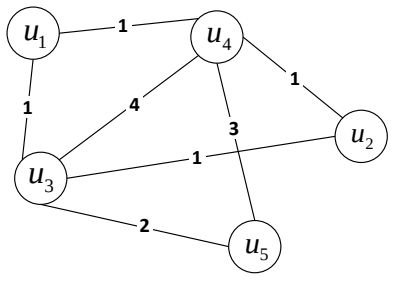


Fig. 2: Shared-Event User Graph

We extend Algorithm 2 to compute partners using the SEUG instead of the EML. Given a kEP query, the algorithm fetches the partner list of the query user. For event e, the

TABLE 7: Adjacency Lists of SEUG

User	Partner List
u_1	$(u_3, 1), (u_4, 1)$
u_2	$(u_3, 1), (u_4, 1)$
u_3	$(u_4, 4), (u_1, 1), (u_2, 1)$
u_4	$(u_3, 4), (u_5, 3), (u_1, 1), (u_2, 1)$
u_5	$(u_4, 3), (u_3, 2)$

users in the partner list are paired sequentially with e, the together preference is computed, and the event-partner pair having the highest together preference at the moment is maintained as the candidate pair. In addition, an upper bound p_r^* (Lemma 4) on the together preference of the remaining users in the partner list is derived. If the together preference of the current candidate pair is no less than p_r^*, the candidate pair is returned (Pruning Rule 3). Otherwise, the algorithm continues to pair the next user with e until Pruning Rule 3 is satisfied.

Lemma 4. *Given a query user u_q and an event e, let (u, num) be the next entry from the partner list of u_q, and define $x = \min\{num, |N(e)|\}$. Then set $TopX(u_q, e)$ contains the x events $\{e_i\}$ in neighborhood $N(e)$ with the largest similarity $s(e, e_i)$. An upper bound on the together preference of the users in the partner list of u_q is*

$$p_r^* = \frac{\sum_{e_i \in TopX(u_q,e)} s(e, e_i)}{\sum_{e_i \in N(e)} s(e, e_i)} \cdot \frac{num}{|E_{u_q}|}. \quad (5)$$

Proof. Since the users in the partner list of u_q are ordered descendingly on their shared events, num is an upper bound on the number of shared events between u_q and the users after u. Then, for each user u_i after u in the partner list, we have $num/|E_{u_q}| \geq J(E_{u_q}, E_{u_i})$. Following Lemma 3, no user in the partner list has attended more than x events in $N(e)$, so set $TopX(u_q, e)$ contains the x events $\{e_i\}$ in neighborhood $N(e)$ with the largest similarity $s(e, e_i)$. Then, for each user u_i after u in the partner list, we have $p_r^* \geq p(u_q, e, u_i)$. □

Pruning Rule 3. *In the extended Algorithm 2, let p_1 be the together preference of the current candidate partner u_p. If $p_1 \geq p_r^*$, user u_p is returned as the partner of event e, and no other user in the fetched member lists can be the partner and can be pruned.*

Example 6. *Given query user u_4 and event e_2, according to Table 4, neighborhood $N(u_4, e_2) = \{e_3, e_5\}$. According to Table 7, $(u_3, 4)$ is first fetched from the partner list of u_4. The together preference $p(u_4, e_2, u_3) = ((0.3 + 0.5)/(0.3 + 0.5)) \cdot (4/5) = 0.8$. User u_3 is taken as the candidate partner. Next, $(u_5, 3)$ is fetched. We have $num = 3$ and $p_r^* = ((0.3 + 0.5)/(0.3 + 0.5)) \cdot (3/5) = 0.6$. Since $p(u_4, e_2, u_3) > p_r^*$, no other user can have higher together preference than does user u_3. Finally, user u_3 is returned as the partner for u_4 at e_2.*

EML + SEUG. When using the EML, the users are retrieved in descending order of the number n_1 of events they have attended. When using the SEUG, the users are retrieved in descending order of the number n_2 of shared events with the query user. We combine these two orders by retrieving users in descending order of $n_1 \cdot n_2$. This scoring predicate is relevant to both the two elements in the together preference and may result in more efficient computation. The empirical study confirms this intuition.

Unpromising-Event Pruning + SEUG. In Section 5.1, an event is pruned if $p_w(e) > p_{ub}(e)$. When using the SEUG, the variable num in Lemma 4 can be used to replace the variable r in $p_{ub}(e)$ (Definition 2), so that the new $p_{ub}(e)$ is a tighter upper bound on the together preference, which can help pruning more events.

6 Empirical Study

6.1 Data and Queries

We have crawled a data set from Meetup[3] that contains 224,238 events and 7,822,965 users. The average number of members per event is 116. We have also downloaded the text descriptions (documents) of the events. The number of unique terms in the document collection is 519,885, and the average number of tokens per document is 72.

The proposed model (Equation 1) is evaluated in a simulated event-partner retrieval scenario. Subsets of the data are extracted, based on which a ground truth is constructed. We compute the top-k event-partner pairs for a target user and validate the result using the ground truth. Specifically, following the idea of time series cross-validation, we sort the events in the data in chronological order and extract 4 subsets, namely the bottom 40%, 30%, 20%, and 10% of the data, as validation sets.

For each validation set, we generate 5 query sets, in which the number of keywords is 1, 2, 3, 4, and 5, respectively. Each query set comprises 100 queries. Specifically, to generate a query, we randomly pick a user u_q as the query user, and we randomly choose words from the document of a randomly selected event e attended by u_q as the query keywords. We ensure that no query has an empty result. The selected pairs (u_q, e) are also used to construct the ground-truth of the event-partner retrieval as follows. For each pair (u_q, e), if u_q and another user u have attended e in the validation set, they are considered as partners of each other w.r.t. event e, denoted as (u_q, e, u). This way, we obtain the ground-truth set $\mathcal{Y} = \{(u_q, e, u)\}$.

6.2 Setup

All algorithms were implemented in Java, and a machine with an Intel(R) Xeon(R) CPU E5-2630 v2@2.60GHz and 128 GB main memory was used for the experiments. The document inverted index is implemented by Lucene[4] and is disk resident. The text relevance of the events is measured by Okapi BM25 [49]. The user-event graph G is represented by adjacency lists and is stored in main memory. Since the structure of the member lists of the events is similar to the adjacency lists of the user-event graph, we extend the adjacency lists of the event nodes in G to include the number of events attended by each user, so that the member list of any event can be obtained from the user-event graph.

Following an existing study [14], we use $Accuracy@n$ to evaluate the event-partner retrieval model. For each positive triple (u_q, e, u) in the ground-truth set $\mathcal{Y}$, we fix (u_q, e) and randomly select 500 users from $U - U_e$ to replace u and

3. http://www.meetup.com
4. https://lucene.apache.org

form a set of negative triples. Similarly, we fix (u_q, u) and randomly select 500 events from $E - (E_{u_q} \cap E_u)$ to replace e and form another set of negative triples. Then, a score is calculated for each triple in $\mathcal{Y}$ and the 1,000 negative triples using the event-partner retrieval model (Equation 1). By ranking the positive and negative triples in descending order of their scores, if the rank of a positive triple is not larger than n, we have a hit. Otherwise, we have a miss. $Accuracy@n$ is defined as follows.

$$Accuracy@n = \frac{\#hit@n}{|\mathcal{Y}|}, \tag{6}$$

where $\#hit@n$ is the total number of hits.

In the experiments, we compare with a previous proposal [20]. In that proposal, together preferences are computed using an existing method [13], while we propose a new way of computing together preferences (cf. Equation 2). In the following experiments, the previous method is denoted by kEP, and the method proposed in this paper is denoted by kEP*. We compare these two methods when varying different parameters. Table 8 shows the parameters and values used in the experiments, where the values in bold are default values.

TABLE 8: Parameter Settings

Parameter	Values
Percentage of validation sets	**10%**, 20%, 30%, 40%
Number of requested event-partner pairs k	1, 5, **10**, 15, 20
Number of query keywords	1, 2, **3**, 4, 5
Event similarity threshold τ	**0.1**, 0.15, 0.2
Parameter α in Equation 1	0.1, **0.2**, 0.3, 0.5, 0.7, 0.9

6.3 Event-Partner Retrieval Model Evaluation

6.3.1 Varying α

In the scoring function (Equation 1), parameter α is used to specify the relative importance of the text relevance of a retrieved event w.r.t. the query keywords versus the probability of the query user and a partner joining an event together. A small α favors the together preference, while a large α favors the text relevance. For the retrieved event-partner pairs of a query, we use $Accuracy@10$ to evaluate the willingness of a query user to join an event with a partner, and to compute the average text relevance of the retrieved events. Table 9 shows the results of kEP and kEP* when varying α. The last column in Table 9 is the average text relevance of the retrieved events and the corresponding rank of kEP*. Taking 11.51 (top-36) as an example, if ranking all the events according to their text relevance w.r.t. the query keywords, the value of text relevance 11.51 is at the top-36 position. As α increases, for both methods, the $Accuracy@10$ decrease while the average text relevance of the retrieved events increases, which is expected. This occurs because when the event-partner pairs are ranked more based on the together preferences, the ranks of the triples with large together preference but small text relevance may be below top-10. On the other hand, if the event-partner pairs are ranked more based on text relevance, the ranks of the triples with small together preference but large text relevance may be below top-10. For most values of α, kEP* outperforms kEP by roughly 50%. The kEP* query aims to retrieve event-partner pairs such that the query user is willing to join the events with the partners and the events are relevant to the query keywords. Based on this intuition, $\alpha = 0.2$ achieves a good $Accuracy@10$ and average text relevance. We thus use this value in the remaining experiments.

TABLE 9: $Accuracy@10$ When Varying α

α	kEP	kEP*	Average textual relevance
0.1	0.398	0.778	10.79 (top-59)
0.2	0.379	0.717	11.51 (top-36)
0.3	0.360	0.656	12.37 (top-21)
0.5	0.322	0.534	14.27 (top-8)
0.7	0.285	0.412	15.72 (top-4)
0.9	0.247	0.290	16.20 (top-4)

6.3.2 Varying the Percentage of Validation Sets

We extract 4 subsets of the data as validation sets. Table 10 shows the $Accuracy@10$ of kEP and kEP* when varying the percentage of validation sets. As expected, the smaller the validation set, the higher the $Accuracy@10$. This is because having more historical information helps determine the relationships among users and events. Further, kEP* also beats kEP in this experiment.

TABLE 10: Varying the Percentage of Validation Sets

Percentage	kEP ($Accuracy@10$)	kEP* ($Accuracy@10$)
40%	0.342	0.669
30%	0.358	0.696
20%	0.364	0.696
10%	0.379	0.717

6.3.3 Varying k

Table 11 shows the $Accuracy@k$ of kEP and kEP* when varying k. As k increases, the $Accuracy@k$ also increases. The reason is that a positive triple has better chance to be ranked within the top-k for large k than for small k. In this experiment, kEP* outperforms kEP significantly for all values of k.

TABLE 11: Varying k

k	kEP ($Accuracy@k$)	kEP* ($Accuracy@k$)
1	0.176	0.338
5	0.337	0.629
10	0.379	0.717
15	0.396	0.749
20	0.401	0.768

6.3.4 Varying the number of query keywords

Table 12 shows the $Accuracy@10$ of kEP and kEP* when varying the number of query keywords. As the number of keywords increases, the $Accuracy@10$ increases slightly. Few number of keywords (e.g., 1 or 2) get more events involved into the computation than many number of keywords (e.g., 4 or 5). The chance of a positive triple to be ranked within the top-k is worse for few number of keywords. Again, kEP* outperforms kEP significantly.

TABLE 12: $Accuracy@10$ When Varying $|\psi_q|$

$\|\psi_q\|$	kEP	kEP*	Average textual relevance
1	0.355	0.694	9.90 (>top-100)
2	0.374	0.712	10.93 (top-55)
3	0.379	0.717	11.51 (top-36)
4	0.381	0.719	12.08 (top-25)
5	0.381	0.719	12.49 (top-19)

6.3.5 Varying document similarity threshold τ

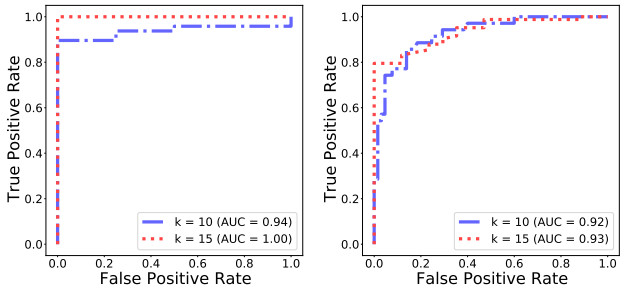
The document similarity is measured using cosine similarity. The maximum cosine similarity in the data set is roughly 0.2. Hence, we vary τ in the range (0, 0.2]. Table 13 shows the $Accuracy@10$ of kEP and kEP* for various value of τ. The $Accuracy@10$ increases as τ increases. When computing the together preference for triple (u_q, e, u), a small τ gets some events with low similarity to e involved. Then, the probability of u_q attending the retrieved event-partner pairs is lower. In this experiment, kEP* outperforms kEP by roughly 50%.

TABLE 13: Varying Document Similarity Threshold

τ	kEP ($Accuracy@10$)	kEP* ($Accuracy@10$)
0.1	0.379	0.717
0.15	0.385	0.725
0.2	0.408	0.792

6.3.6 ROC-Curve based Evaluation

In order to evaluate the proposed method using the ROC curve, we convert the kEP problem into a binary classification problem. Using $Accuracy@k$ as defined in Equation 6 mentioned in Section 6.2, we construct 100 testing samples. Each sample consists of 1 positive triple and 1000 negative triples. Let r be the average rank of the positive triple in one sample. If $r \leq k$, the label of this sample is "1". Otherwise, the label of this sample is "0". The $Accuracy@k$ of each sample is the classification probability. Figure 3 shows the ROC curves of kEP* and kEP when $k = 10$ and $k = 15$. The areas under the ROC curves of kEP* are larger than those of kEP.



(a) kEP* (b) kEP

Fig. 3: ROC Curves

6.4 Efficiency Evaluation

We proceed to cover a study of the performance of the proposed framework and its three optimizations.

6.4.1 Methods Compared and Parameter Default Values

In the experiments, F-NLJ denotes the framework adopting the nested loop join without optimizations. Further, we let F-NLJ^1, F-NLJ^2, and F-NLJ^3 denote the approach with the optimization technique proposed in Section 5.1, Section 5.2, and Section 5.3, respectively. Next, we let F-NLJ* denote the framework adopting the nested loop join with all three optimizations. Similarly, we use F-RJ, F-RJ^1, F-RJ^2, F-RJ^3, and F-RJ* to represent the approaches using the ripple join strategy.

We study the efficiency of the proposed algorithms when varying a single parameter and keeping the remaining parameters at default values that are given as follows: the number k of requested event-partner pairs is 10; the number of query keywords is 3; parameter α in the scoring function (Equation 1) is set to 0.2. Some queries take a long time to compute using the framework without optimizations. If the processing of any query exceeds 20 seconds, we stop the processing.

In each experiment, for each algorithm, we report the average runtime, the average number of computed events per query, the average number of pruned events, and the average number of computed users per query.

6.4.2 Computational Complexity

In the kEP problem, the number of users is $|U|$, and the number of events is $|E|$. The complexity of the kEP problem is $O(|U| \cdot |E|)$, which means that in the computation, $|E|$ events are involved and that for each event, $|U|$ users are considered.

The proposed algorithm reduces the computational cost mainly with respect to the following two aspects:

- Two techniques are included in the algorithm that aim to reduce the number of events $|E|$ that need to be considered in computations, i.e., threshold T in Algorithm 1 and the unpromising-event pruning covered in Section 5.1.
- Two techniques are included in the algorithm to reduce the number of users $|U|$ that need to be considered in computations, i.e., the partner computation based on the EML covered in Section 5.2 and the partner computation based on the SEUG covered in Section 5.3.

The number of events that need to be considered depends on the distribution of the textual relevance of the events, $t(\psi_q, e)$, the distribution of the document similarity between events, $s(e, e_i)$, and the distribution of the number of events attended by both the query user and by other users. The number of users that need to be considered depends on the distribution of the document similarity between events, $s(e, e_i)$, the distribution of the number of events attended by both the query user and by other users, and the distribution of the number of events attended by users.

Algorithm $\mathcal{A}_0$ [50] is a general framework for returning the top-k answers for a monotone query by aggregating several sorted lists. Fagin derives the cost of algorithm $\mathcal{A}_0$ to be $O(N^{(m-1)/m}k^{1/m})$, where m is the number of aggregated sorted lists and k is the number of requested results. In the

proposed algorithm, given $N = \max\{|U|, |E|\}$, $m = 2$, and $k = 1$, the cost is $O(N^{\frac{1}{2}})$.

In real datasets, on average, many events have low textual relevance to query keywords, many event pairs have low document similarity, and many users attend few events (compared to $|E|$). Hence, the proposed algorithm on average is able to disregard large numbers of events and users in computations. To quantify the performance improvements, Table 14 compares the complexity of the kEP problem and the proposed algorithms F-NLJ* and F-RJ* in terms of the number of events and users considered in computations on the real dataset used in the experiments. We report the fractions of events and users considered by the algorithm for different values of parameters α, k, and the number of keywords. Each row shows the fractions when one parameter has been changed while the other two parameters take the default values shown in Table 8. Both F-NLJ* and F-RJ* consider six orders of magnitude fewer pairs that the worst case.

TABLE 14: Complexity Comparision

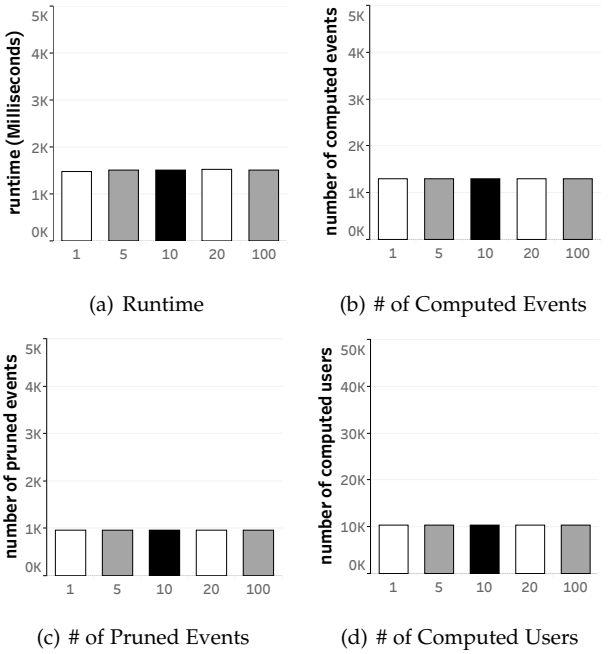
Parameters	F-NLJ*	F-RJ*
$\alpha = 0.1$	$4.71‰\|E\| \cdot 2.11‰\|U\|$	$4.74‰\|E\| \cdot 2.04‰\|U\|$
$\alpha = 0.5$	$2.52‰\|E\| \cdot 0.43‰\|U\|$	$2.61‰\|E\| \cdot 0.49‰\|U\|$
$\alpha = 0.9$	$1.83‰\|E\| \cdot 0.07‰\|U\|$	$2‰\|E\| \cdot 0.16‰\|U\|$
$k = 1$	$2.18‰\|E\| \cdot 0.3‰\|U\|$	$2.35‰\|E\| \cdot 0.29‰\|U\|$
$k = 10$	$3.6‰\|E\| \cdot 1.32‰\|U\|$	$3.66‰\|E\| \cdot 1.32‰\|U\|$
$k = 20$	$4.05‰\|E\| \cdot 1.89‰\|U\|$	$4.11‰\|E\| \cdot 1.89‰\|U\|$
$\#\text{keywords} = 1$	$1.51‰\|E\| \cdot 0.91‰\|U\|$	$1.54‰\|E\| \cdot 0.86‰\|U\|$
$\#\text{keywords} = 3$	$3.6‰\|E\| \cdot 1.32‰\|U\|$	$3.66‰\|E\| \cdot 1.32‰\|U\|$
$\#\text{keywords} = 5$	$5.05‰\|E\| \cdot 1.38‰\|U\|$	$5.1‰\|E\| \cdot 1.37‰\|U\|$

6.4.3 Tuning the Number of Fetched Events and Users in the Ripple Join

This experiment studies the effect of the numbers of fetched events and users at a time in the ripple join on performance. We thus vary the numbers of fetched events and users from 1 to 100. Figure 4 shows that the performance is insensitive to the number of fetched events and users. Thus, in the following experiments, the numbers of fetched events and users in the ripple join are both set to 1.

6.4.4 Evaluation of Optimizations

Figures 5 and 6 show the performance of the proposed optimizations when using the nested loop join and the ripple join, respectively. F-NLJ^1 and F-RJ^1 adopt the unpromising-event pruning proposed in Section 5.1. Figures 5(c) and 6(c) show that this optimization technique reduces the number of events computed during the processing of the kEP query. F-NLJ^2 and F-RJ^2 find partners for events using the EML (introduced in Section 5.2). F-NLJ^3 and F-RJ^3 compute partners for events using the SEUG (presented in Section 5.3). These two optimization techniques both aim for efficient partner computation. Figures 5(d) and 6(d) show that these two optimizations reduce the number of users computed for each query on average. F-NLJ* and F-RJ* are the algorithms that combine all three optimization techniques. As shown in Figures 5 and 6, F-NLJ* and F-RJ* significantly outperform F-NLJ and F-RJ that are the algorithms without any optimization. Overall, the proposed optimizations speed up the processing of the kEP query. The performance of the

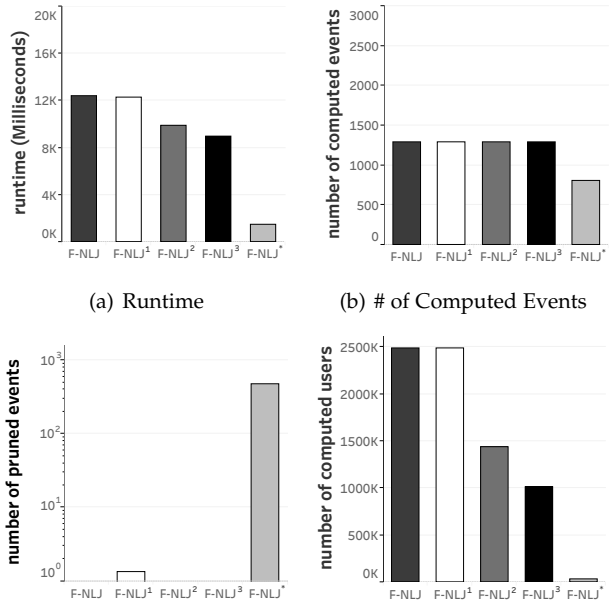


(a) Runtime (b) # of Computed Events

(c) # of Pruned Events (d) # of Computed Users

Fig. 4: Varying the Number of Fetched Users in Ripple Join

algorithms using the ripple join is better than that using the nested loop join. The reason is that the ripple join is designed to avoid complete data scans and has been shown more efficiently than the nested loop join [19]. In the remaining experiments, we only consider F-NLJ, F-NLJ*, and F-RJ*.



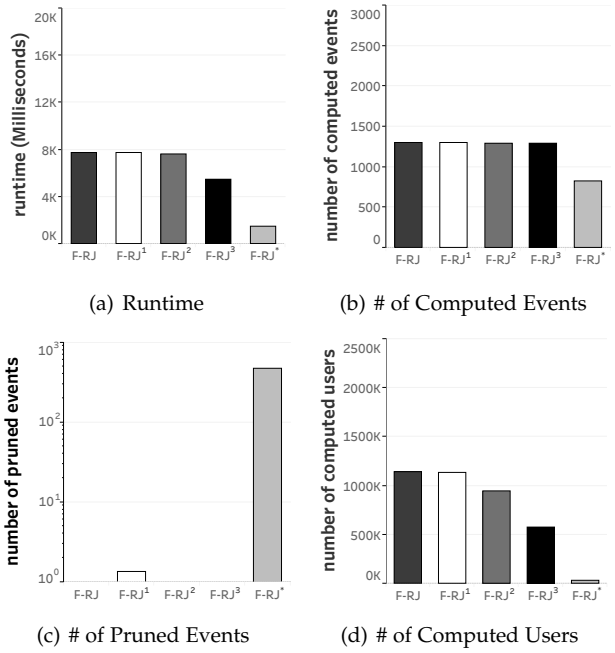
(a) Runtime (b) # of Computed Events

(c) # of Pruned Events (d) # of Computed Users

Fig. 5: Performance of Optimizations Using Nested Loop Join

6.4.5 Varying the Number k of Requested Event-Partner Pairs

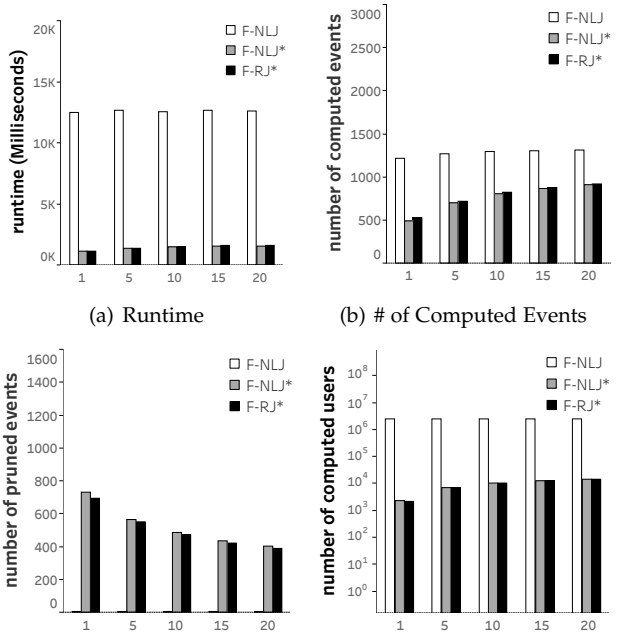
Figure 7 shows the performance of the evaluated algorithms when varying k. The average runtime of the three



(a) Runtime (b) # of Computed Events

(c) # of Pruned Events (d) # of Computed Users

Fig. 6: Performance of Optimizations Using Ripple Join



(a) Runtime (b) # of Computed Events

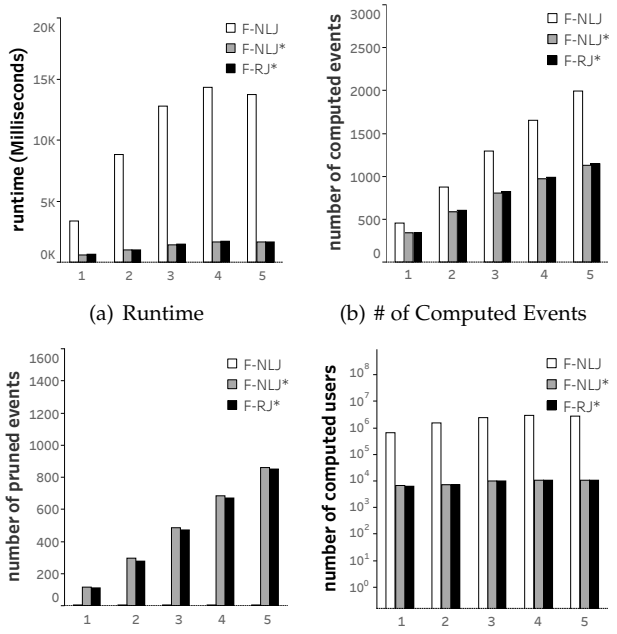
(c) # of Pruned Events (d) # of Computed Users

Fig. 7: Varying the Number of Requested Event-Partner Pairs

approaches increases slightly as k increases, since more relevant events are retrieved and more users are involved. The unpromising event pruning optimization prunes many events, as shown in Figure 7(c). The efficient partner computation optimizations reduce the number of retrieved users per query, as shown in Figure 7(d). F-NLJ retrieves more events than do F-NLJ* and F-RJ* (Figure 7(b)). This is because F-NLJ* and F-RJ* have tighter upper bounds on the together preferences. Thus, the framework with optimizations outperforms F-NLJ significantly in terms of runtime. F-RJ* has slightly fewer pruned events than does F-NLJ*, but it has slightly fewer retrieved users per event than does F-NLJ*. Thus, the runtimes of F-RJ* and F-NLJ* are similar.

6.4.6 Varying the Number of Query Keywords

Figure 8 shows the performance of the evaluated algorithms when varying the number of query keywords. It can be seen that also here, the unpromising event pruning and efficient partner computation optimizations are effective, cf. Figures 8(c) and 8(d). The runtime shows an increasing trend as the number of query keywords increases. This is because more query keywords results in more events being involved in the computation. Overall, F-NLJ* and F-RJ* outperform F-NLJ significantly in terms of runtime.



(a) Runtime (b) # of Computed Events

(c) # of Pruned Events (d) # of Computed Users

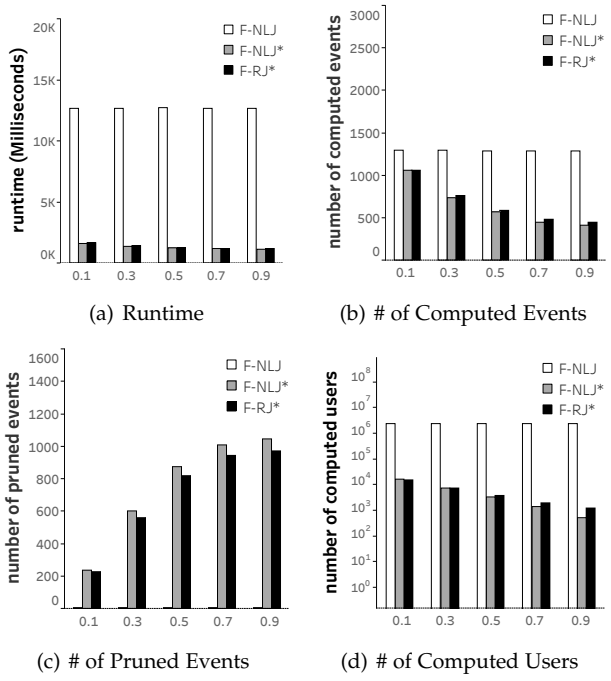
Fig. 8: Varying the Number of Keywords

6.4.7 Varying α

Figure 9 reports on the findings when varying α that controls the weight of the textual relevance in the scoring function. The three approaches perform slightly better (shorter runtime, fewer retrieved events and users) as α increases. The reason is that a large α gives high weight to the textual relevance, so that the ranking of the event-partner pairs is affected more by the textual relevance of the events than by the together preferences. Since events are retrieved in descending order of the textual relevance in the three approaches, the top-k event-partner pairs are determined faster when α is large. Consistent with the previous results, this experiment shows that the proposed optimizations are effective, i.e, many unpromising events are pruned, and the number of retrieved users per query is reduced.

6.4.8 Summary

Overall, for a broad range of parameter settings, the proposed optimizations improve the performance of the framework substantially. Unpromising events are pruned. The numbers of users needed for finding partners for events are



(a) Runtime (b) # of Computed Events

(c) # of Pruned Events (d) # of Computed Users

Fig. 9: Varying α

reduced. In most cases, the ripple join and the nested loop join perform similarly.

7 Conclusion

This paper defines and enables efficient computation of the top-k event-partner (kEP) pair retrieval query that accommodates both event-partner recommendation and keyword-based search. Given a query user, keywords, and a number k, the query retrieves event-partner pairs from a bipartite event-user graph, where events have text descriptions, taking into account both the textual relevance of events to the query keywords and so-called together preferences that capture how much the query user prefers to attend an event with a particular partner. A rank-join based framework is proposed for computing this query. To improve efficiency, the framework comes with three optimizations. The paper's empirical study offers insight into the proposed techniques, indicating that they are effective and that the framework is practical.

The paper's study opens to a number of promising directions for future studies. First, it is of interest to understand how kEP queries can be best processed when the query user's current location is taken into account. Second, how to measure together preferences is an interesting problem. This paper considers the relationship between one partner and the query user. It is also of interest to investigate the effect of collectiveness [51] when a query user belongs to a group of partners.

Acknowledgement

This work was supported in part by grants No. 2019A1515011721 and No. 2019A1515011064 from Natural Science Foundation of Guangdong Province, China and grant No. 20200806102941001 from Shenzhen Fundamental Research Program, China.

References

[1] L. Gao, J. Wu, Z. Qiao, C. Zhou, H. Yang, and Y. Hu, "Collaborative social group influence for event recommendation," in *CIKM*, 2016, pp. 1941–1944.

[2] W. Zhang and J. Wang, "A collective Bayesian Poisson factorization model for cold-start local event recommendation," in *KDD*, 2015, pp. 1455–1464.

[3] X. Ji, Z. Qiao, M. Xu, P. Zhang, C. Zhou, and L. Guo, "Online event recommendation for event-based social networks," in *WWW*, 2015, pp. 45–46.

[4] Z. Qiao, P. Zhang, C. Zhou, Y. Cao, L. Guo, and Y. Zhang, "Event recommendation in event-based social networks," in *AAAI*, 2014, pp. 3130–3131.

[5] Z. Qiao, P. Zhang, Y. Cao, C. Zhou, L. Guo, and B. Fang, "Combining heterogenous social and geographical information for event recommendation," in *AAAI*, 2014, pp. 145–151.

[6] J. Manotumruksa, C. MacDonald, and I. Ounis, "Regularising factorised models for venue recommendation using friends and their comments," in *CIKM*, 2016, pp. 1981–1984.

[7] H. Li, Y. Ge, R. Hong, and H. Zhu, "Point-of-interest recommendations: Learning potential check-ins from friends," in *KDD*, 2016, pp. 975–984.

[8] X. Chen, Y. Zeng, G. Cong, S. Qin, Y. Xiang, and Y. Dai, "On information coverage for location category based point-of-interest recommendation," in *AAAI*, 2015, pp. 37–43.

[9] D. Lian, C. Zhao, X. Xie, G. Sun, E. Chen, and Y. Rui, "GeoMF: Joint geographical modeling and matrix factorization for point-of-interest recommendation," in *KDD*, 2014, pp. 831–840.

[10] B. Liu, Y. Fu, Z. Yao, and H. Xiong, "Learning geographical preferences for point-of-interest recommendation," in *KDD*, 2013, pp. 1043–1051.

[11] S. Wan, Y. Lan, J. Guo, C. Fan, and X. Cheng, "Informational friend recommendation in social media," in *SIGIR*, 2013, pp. 1045–1048.

[12] Y. Lu, Z. Qiao, C. Zhou, Y. Hu, and L. Guo, "Location-aware friend recommendation in event-based social networks: A bayesian latent factor approach," in *CIKM*, 2016, pp. 1957–1960.

[13] W. Tu, D. W. Cheung, N. Mamoulis, M. Yang, and Z. Lu, "Activity recommendation with partners," *ACM Trans. Web*, vol. 12, no. 1, pp. 4:1–4:29, 2018.

[14] H. Yin, L. Zou, Q. V. H. Nguyen, Z. Huang, and X. Zhou, "Joint event-partner recommendation in event-based social networks," in *ICDE*, 2018, pp. 929–940.

[15] M. Sharma, J. Zhou, J. Hu, and G. Karypis, "Feature-based factorized bilinear similarity model for cold-start top-*n* item recommendation," in *SIAM*, 2015, pp. 190–198.

[16] H. Chen and P. Chen, "Differentiating regularization weights - A simple mechanism to alleviate cold start in recommender systems," *ACM Trans. Knowl. Discov. Data*, vol. 13, no. 1, pp. 8:1–8:22, 2019.

[17] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *SIGIR*, 1998, pp. 275–281.

[18] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid, "Supporting top-k join queries in relational databases," *VLDB J.*, vol. 13, no. 3, pp. 207–221, 2004.

[19] P. J. Haas and J. M. Hellerstein, "Ripple joins for online aggregation," in *SIGMOD*, 1999, pp. 287–298.

[20] D. Wu, Y. Zhu, and C. S. Jensen, "In good company: Efficient retrieval of the top-k most relevant event-partner pairs," in *DASFAA*, 2019, pp. 519–535.

[21] T. A. N. Pham, X. Li, G. Cong, and Z. Zhang, "A general graph-based model for recommendation in event-based social networks," in *ICDE*, 2015, pp. 567–578.

[22] H. Yin, Y. Sun, B. Cui, Z. Hu, and L. Chen, "LCARS: A location-content-aware recommender system," in *KDD*, 2013, pp. 221–229.

[23] A. Q. Macedo, L. B. Marinho, and R. L. Santos, "Context-aware event recommendation in event-based social networks," in *RecSys*, 2015, pp. 123–130.

[24] J. Yu, M. Gao, H. Yin, J. Li, C. Gao, and Q. Wang, "Generating reliable friends via adversarial training to improve social recommendation," in *ICDM*, 2019, pp. 768–777.

[25] J. Gorla, N. Lathia, S. Robertson, and J. Wang, "Probabilistic group recommendation via information matching," in *WWW*, 2013, pp. 495–504.

[26] Q. Yuan, G. Cong, and C. Lin, "COM: a generative model for group recommendation," in *KDD*, 2014, pp. 163–172.

[27] H. Yin, Q. Wang, K. Zheng, Z. Li, J. Yang, and X. Zhou, "Social influence-based group representation learning for group recommendation," in *ICDE*. IEEE, 2019, pp. 566–577.

[28] L. Guo, H. Yin, Q. Wang, B. Cui, Z. Huang, and L. Cui, "Group recommendation with latent voting mechanism," in *ICDE*. IEEE, 2020, pp. 121–132.

[29] Y. Du, X. Meng, Y. Zhang, and P. Lv, "Gerf: A group event recommendation framework based on learning-to-rank," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 4, pp. 674–687, 2020.

[30] Y. Du, X. Meng, and Y. Zhang, "Cvtm: A content-venue-aware topic model for group event recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 7, pp. 1290–1303, 2020.

[31] A. Natsev, Y. Chang, J. R. Smith, C. Li, and J. S. Vitter, "Supporting incremental join queries on ranked inputs," in *VLDB*, 2001, pp. 281–290.

[32] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid, "Joining ranked inputs in practice," in *VLDB*, 2002, pp. 950–961.

[33] I. F. Ilyas, W. G. Aref, A. K. Elmagarmid, H. G. Elmongui, R. Shah, and J. S. Vitter, "Adaptive rank-aware query optimization in relational databases," *ACM Trans. Database Syst.*, vol. 31, no. 4, pp. 1257–1304, 2006.

[34] C. Li, K. C. Chang, I. F. Ilyas, and S. Song, "RankSQL: Query algebra and optimization for relational top-k queries," in *SIGMOD*, 2005, pp. 131–142.

[35] N. Mamoulis, M. L. Yiu, K. H. Cheng, and D. W. Cheung, "Efficient top-k aggregation of ranked inputs," *ACM Trans. Database Syst.*, vol. 32, no. 3, p. 19, 2007.

[36] J. Finger and N. Polyzotis, "Robust and efficient algorithms for rank join evaluation," in *SIGMOD*, 2009, pp. 415–428.

[37] K. Schnaitter and N. Polyzotis, "Optimal algorithms for evaluating rank joins in database systems," *ACM Trans. Database Syst.*, vol. 35, no. 1, pp. 6:1–6:47, 2010.

[38] A. Al-Baghdadi and X. Lian, "Topic-based community search over spatial-social networks," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2104–2117, 2020.

[39] L. Chen, C. Liu, R. Zhou, J. Li, X. Yang, and B. Wang, "Maximum co-located community search in large scale social networks," *Proc. VLDB Endow.*, vol. 11, no. 10, pp. 1233–1246, 2018.

[40] F. Guo, Y. Yuan, G. Wang, X. Zhao, and H. Sun, "Multi-attributed community search in road-social networks," in *ICDE*, 2021, pp. 109–120.

[41] Y. Kou, D. Shen, Q. Snell, D. Li, T. Nie, G. Yu, and S. Ma, "Efficient team formation in social networks based on constrained pattern graph," in *ICDE*, 2020, pp. 889–900.

[42] Q. Li, Y. Zhu, and J. X. Yu, "Skyline cohesive group queries in large road-social networks," in *ICDE*, 2020, pp. 397–408.

[43] L. Chen, C. Liu, K. Liao, J. Li, and R. Zhou, "Contextual community search over large social networks," in *ICDE*, 2019, pp. 88–99.

[44] F. Guo, Y. Yuan, G. Wang, L. Chen, X. Lian, and Z. Wang, "Cohesive group nearest neighbor queries over road-social networks," in *ICDE*, 2019, pp. 434–445.

[45] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, Nov. 1975.

[46] S. J. Fitz-Gerald and B. Wiggins, "Introduction to modern information retrieval, 3rd ed., C.G. chowdhury. facit publishing, london (2010)," *Int. J. Inf. Manag.*, vol. 30, no. 6, pp. 573–574, 2010.

[47] P. Tan, M. S. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison-Wesley, 2005.

[48] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *J. Comput. Syst. Sci.*, vol. 66, no. 4, pp. 614–656, 2003.

[49] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. USA: Cambridge University Press, 2008.

[50] R. Fagin, "Combining fuzzy information from multiple systems," in *PODS*, 1996, pp. 216–226.

[51] X. Li, M. Chen, and Q. Wang, "Measuring collectiveness via refined topological similarity," *ACM Trans. Multim. Comput. Commun. Appl.*, vol. 12, no. 2, pp. 34:1–34:22, 2016.

Dingming Wu is Associate Professor of College of Computer Science & Software Engineering at Shenzhen University, China. She received her Bachelor degree in Computer Science at Huazhong University of Science and Technology, Wuhan, China in 2005, and a Master degree in Computer Science at Peking University, Beijing, China in 2008. She received a Ph.D. degree in Computer Science at Aalborg University, Denmark in 2011. Her research concerns data management, query processing, information retrieval, and data mining.

Erjia Xiao received the bachelor's degree from the College of Computer Science & Software Engineering, Shenzhen University, in 2020. His research interests include social networking data mining and information retrieval.

Yi Zhu received the bachelor's degree from Shenzhen University in 2016 and the Master degree in Computer Science from Shenzhen University in 2019. Her research interests include social networking data mining and information retrieval.

Christian S. Jensen is Professor of Computer Science at Aalborg University, Denmark. His research concerns analytics, data management, and data-intensive systems, and it centers around temporal and spatio-temporal analytics, including machine learning, data mining, and query processing. Christian is an ACM and IEEE Fellow, and he is a member of Academia Europaea, the Royal Danish Academy of Sciences and Letters, and the Danish Academy of Technical Sciences. He has received several awards for his research, most recently the 2019 IEEE TCDE Impact Award.

Kezhong Lu is Professor of College of Computer Science & Software Engineering at Shenzhen University, China. He received his Bachelor degree and Ph.D. degree in Computer Science at University of Science and Technology of China in 2001 and 2006. His research concerns big data, parallel and distributed computing, algorithms, wireless sensor network, and computational geometry. He is a vice dean of College of Computer Science & Software Engineering and the leader of computer technology area at Shenzhen University, China.