In Good Company: Efficient Retrieval of the Top-*k* Most Relevant Event-Partner Pairs

Dingming Wu[1](✉), Yi Zhu[1], and Christian S. Jensen[2]

[1] College of Computer Science and Software Engineering,
Shenzhen University, Shenzhen, China
dingming@szu.edu.cn, zhuyi2016@email.szu.edu.cn

[2] Department of Computer Science, Aalborg University, Aalborg, Denmark
csj@cs.aau.dk

Abstract. The proliferation of event-based social networking (ESBN) motivates a range of studies on topics such as event, venue, and friend recommendation and event creation and organization. In this setting, the notion of event-partner recommendation has recently attracted attention. When recommending an event to a user, this functionality allows recommendation of partner with whom to attend the event. However, existing proposals are push-based: 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 push and pull, 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 the top-k event-partner (kEP) pair retrieval query that integrates event-partner recommendation and keyword-based search for events. The 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. In order 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.

1 Introduction

The recent proliferation of the 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 [5,12,23,24,33], venues [2,14–16,20], and friends [17,28,31]. Unlike previous recommendation techniques that each focus on recommending only one type

[1] http://www.meetup.com.

[2] http://www.eventbrite.com.

G. Li et al. (Eds.): DASFAA 2019, LNCS 11447, pp. 519–535, 2019.
https://doi.org/10.1007/978-3-030-18579-4_31

of items (either events or friends), event-partner recommendation [27] aims to recommend events together with partners to users. The rationale is that attending an event with a partner may be more attractive to a user than attending the event alone. Our put differently, a user may not attend an event if the user has to do so alone. However, no matter which recommendation technique is used, users receive information in a push mode, and the provided information is only related to the users' historical data. Yes, EBSNs also provide search services that allow users to retrieve event information in response to query keywords. This way users retrieve information in pull mode according to specified query keywords.

We propose a new way of accessing information in the EBSNs that combines push and pull modes, thus allowing users not only to conduct ad-hoc event search, but also to receive recommended partners for retrieved events. To achieve this goal, one may consider extending existing methods by first recommending events to a user and then filter out irrelevant events according to given keywords. However, this approach may produce empty results, since recommended events are usually based on a user's historical information while given keywords are ad hoc and may not align with the historical data. We adopt a different tack: we first retrieve relevant events w.r.t. given keywords, and then, for each relevant event, we find an appropriate partner.

Hence, we propose a new kind of EBSN query that takes advantage of both recommendation and search techniques. Taking into account user-specified keywords and a user's historical data, it retrieves event-partner pairs, such that the events are relevant to the keywords and such that the query user is willing to attend the events with the suggested partners. For instance, user "Mary" wants to attend a "rock concert". The query we propose retrieves k events relevant to "rock concert" and suggests a partner for each event. This of query is called the *top-k event-partner (kEP) pair retrieval query*. It differs from keyword queries that retrieve relevant events without partners, and is also differs from event-partner recommendation based on only the historical data, where ad-hoc query keywords are not taken into consideration.

The kEP query can be modeled as a top-k join query that returns the k join results (pairs of events and users) with the highest scores. A straightforward method to process the kEP query is to first join events and the users and then, for each event e, choose the pair (e, u_i) with the highest so-called together preference as a result candidate. Next, all the candidates are ranked according to a scoring function, and the k candidates that score the highest are returned as the results. However, this method is inefficient, since all possible combinations of events and users are considered in the join process. One may consider to extend the rank-join algorithm [10] to answer the kEP queries, yielding a method that is more efficient than the straightforward method just described. The idea is to scan input events and users ordered according to their scoring predicates. While a scoring function might use textual relevance of descriptions of events to the query keywords, there is no obvious scoring predicate 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 attended. Intuitively, users who have attended many events tend to have high so-called together preferences, to be detailed in the next section. Two representative join strategies, nested loop join and ripple join [7], are studied within the framework. An empirical study offers evidence that the ripple join is better than the nested loop join. To further improve the performance of the framework, three optimizations are proposed: (1) an unpromising-event pruning technique that removes encountered events that cannot enter the result, (2) a key partner technique that quickly identifies results by exploiting a property of the scoring function, and (3) an efficient partner computation technique that reduces the computational cost of finding the partner with the highest together preference for an event. To evaluate the proposed framework and optimizations, we conduct experiments with prototype implementations of the proposed techniques using real data. The results offer insight into the properties of the techniques and indicate that the paper's proposal is useful in practice.

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

2 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 models a set of events posted by the users, and $R \subseteq U \times E$ 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 will assume that a document is represented by a term vector [25]. The members of an event e are the users u who have joined $e : \{u|(u, e) \in R\}$.

The **together preference** [27] $p(u^*, e, u)$ measures the probability that user u^* is willing to attend event e with user u, $u^* \neq u$, i.e., user u is willing to be a partner of u^* at event e. Its definition is given in Eq. 1, 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.

$$p(u^*, e, u) = \frac{\sum_{e_i \in N(u^*, e)} s(e, e_i) \cdot b(u^*, e_i, u)}{\sum_{e_i \in N(u^*, e)} s(e, e_i)} \qquad (1)$$

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 attend event e with partner user u. In the definition, $N(u^*, e)$ is the neighborhood of a pair of a user and an event (u^*, e), which is the set of events e_i that satisfy the following two conditions: (1) user u^* has attended event e_i, and (2) the similarity $s(e, e_i)$ between the documents of events e and e_i is no less than a threshold τ. We define the similarity $s(e, e_i)$ as the cosine similarity. However, the proposed method is independent of the choice of the similarity measure. Any reasonable similarity function can be adopted easily. Given a partner user u, a target user u^*, and an event e_i, $b(u^*, e_i, u) = 1$ if u^* and u have participated in event e_i, i.e., $(u^*, e_i) \in R$ and $(u, e_i) \in R$; otherwise, $b(u^*, e_i, u) = 0$. The denominator is the sum of the similarities between event e and each event e_i in the neighborhood $N(u^*, e)$. The numerator sums up the similarities between event e and the event e_i in neighborhood $N(u^*, e)$ that u has participated in. The together preference is not symmetric, i.e., $p(u^*, e, u) \neq p(u, e, u^*)$. If $N(u^*, e) = \emptyset$, the together preference $p(u^*, e, u)$ is undefined, which means that no partner can be recommended for u^* for participating in event e. The together preference can be interpreted in two phases. First, an event e is taken as a candidate event for user u^* if some of u^*'s previously attended events are similar to e, i.e, $N(u^*, e) \neq \emptyset$. Second, a user u is considered a candidate partner w.r.t. u^* and e if u has participated in events in neighborhood $N(u^*, e)$.

Example 1. Consider the EBSN in Fig. 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 user-event pair (u_4, e_3), $N(u_4, e_3)$ is $\{e_2, e_4, e_5\}$ because (i) user u_4 has attended events e_1, e_2, e_3, e_4, and e_5, and (ii) the similar (having similarity no less than 0.3) events of e_3 are e_2, e_4, and e_5. The together preference $p(u_4, e_3, u_3) = (0.5 + 0.6)/(0.5 + 0.3 + 0.6) = 0.79$, since u_3 has attended events e_2 and e_5 in neighborhood $N(u_4, e_3)$.

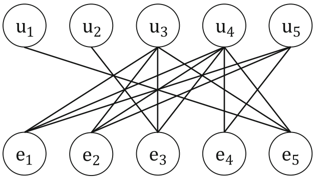


Fig. 1. Example EBSN

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

	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

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 arguments: (i) a number k of requested event-partner pairs, (ii) a query user u_q, and (iii) a set of query keywords ψ_q. Let $t(\psi_q, e.\psi)$ be the textual relevance (e.g., defined using language models [22]) of event e w.r.t. the query keywords ψ_q. For each $e \in E$, let u^e be the user who maximizes

$p(u_q, e, u)$, i.e., $u^e = \arg\max_{u \in U} p(u_q, e, u)$. The result of a kEP query contains k event-partner pairs (e, u) with the highest score $f(u_q, \psi_q, e, u^e)$ (Eq. 2). 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 and the together preference. Ties are broken arbitrarily.

The scoring function used in the paper 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 tries to find 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.

$$\begin{aligned} f(u_q, \psi_q, e, u^e) &= \alpha \cdot t(\psi_q, e.\psi) + (1 - \alpha) \cdot p(u_q, e, u^e) \\ &\qquad s.t.\ \ t(\psi_q, e) \in [0, 1] \quad \wedge \quad p(u_q, e, u^e) \in [0, 1] \end{aligned} \tag{2}$$

In the 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 those events, since the query user is probably interested in them and may attend them again.

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, and the neighborhoods of user-event pairs are shown in Table 4 (given $\tau = 0.3$). Given the query user u_4, for each event e, we choose user u^e as the partner, given that the together preference $p(u_4, e, u^e)$ exceeds the together preference of choosing any other user. Table 4 shows the partner user 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 score 0.85 and 0.8, respectively.

Table 3. Textual relevance

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. Neighborhood

Event	$N(u_4, e)$	u^e	$p(u_4, e, u^e)$
e_1	e_4, e_5	u_5	0.7
e_2	e_3, e_5	u_3	1
e_3	e_2, e_4, e_5	u_3	0.79
e_4	e_1, e_3, e_5	u_3	1
e_5	e_1, e_2, e_3, e_4	u_3	0.8

3 Rank-Join Based Framework

We proceed to present the rank-join based framework for the processing of kEP queries. Section 3.1 presents the main data structures used in the framework. Section 3.2 explains the query processing algorithm, and Sect. 3.3 covers two join strategies in the framework.

3.1 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 the main memory. The other one 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 $e.\psi$.

3.2 Algorithm

Following the idea of the rank-join algorithm [10], 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 (Eq. 1), the framework uses a heuristic scoring predicate of the users, namely the number of events the users have attended. The motivation is that the users who have attended many events can be expected to have high together preferences w.r.t. query user u_q and event e. However, this heuristic falls short when a user has attended many events outside neighborhood $N(u_q, e)$. To avoid this situation, the framework retrieves the users based on two constraints, i.e., (1) the retrieved users should have attended at least one event in neighborhood $N(u_q, e)$, 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 into the join process is constructed as follows. For each retrieved event e, its neighborhood $N(u_q, e)$ is computed. Then, for each event in the neighborhood, its participants are obtained from the event-user graph. The user set $U(u_q, e)$ is the union of the members of all the events in neighborhood $N(u_q, e)$. 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) [3] 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 (Eq. 2). 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 not difficult 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. Different join strategies can be adopted for the join in the algorithm.

3.3 Join Strategies

We consider two state-of-the-art join strategies in the framework, namely nested loop join and ripple join [7]. The nested loop join consists of two nested loops, 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, one previously unseen tuple (user or event) is retrieved from each of the two input lists in each step; these new tuples are joined with the previously seen tuples and with each other. As in the nested loop join, the events are retrieved in the 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.

4 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 optimizations with the goal of improving the performance of the framework.

4.1 Unpromising-Event Pruning

This optimization reduces the computational cost by pruning events that will definitely 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.*

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

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

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

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 Eq. 1, the numerator of the together preference sums up the similarities between the events e_i that user u has attended in neighborhood $N(u_q, e)$. This can be rewritten as follows: $\sum_{e_i \in N(u_q,e) \cap E_u} s(e, e_i)$. Set $TopM(u_q, e)$ contains the top-m similar events in neighborhood $N(u_q, e)$. Since $m = \max_{u \in (U \setminus \{u_q\})}\{|E_c| \mid E_c = N(u_q, e) \cap E_u\}$ is the maximum number of events attended by a user in $N(u_q, e)$, we have $\forall u \in (U \setminus \{u_q\})(|TopM(u_q, e)| \geq |N(u_q, e) \cap E_u|)$. It is easy to derive that $\forall u \in (U \setminus \{u_q\})(\sum_{e_i \in N(u_q,e) \cap E_u} s(e, e_i) \leq \sum_{e_i \in TopM(u_q,e)} s(e, e_i))$. The denominators of $p_{ub}(e)$ and the together preference are the same. Hence, we have proven 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 illustrate how to compute the best possible together preference of event e_3. Neighborhood $N(u_4, e_3) = \{e_2, e_4, e_5\}$ and $m = 2$. Then we have $TopM(u_4, e_3) = \{e_2, e_5\}$. The best possible together preference is calculated as $p_{ub}(e_3) = (0.5 + 0.6)/(0.5 + 0.3 + 0.6) = 0.79$.

Lemmas 1 and 2 present the properties of $p_w(e)$ and $p_{ub}(e)$, respectively. Pruning Rule 1 below is based on Lemmas 1 and 2 and is able to prune unpromising events (that cannot contribute to pairs in the top-k result). Thus it enables reducing 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.*

4.2 Key Partner

It follows from the definition of the together preference (Eq. 1) that if a user u exists who has attended all events in neighborhood $N(u_q, e)$, the together preference $p(u_q, e, u)$ equals 1, the maximum value. Thus, user u is the partner for event e. Based on this observation, we introduce the key partner set (Definition 3) of a user u. If a query user u_q has a key partner, any candidate event for u_q will be paired with the key partner, and the together preference is 1 (Lemma 3). In other words, computing the top-k event-partner pairs for a query user who has a key partner is efficient: the k events with the highest textual relevance are retrieved and paired with the key partner.

Definition 3. ***Key Partner Set*** *$KP(u)$: Let E_u and $E_{u'}$ be the sets of events attended by users u and u', respectively. If $E_u \subseteq E_{u'}$, user u' is a key partner of user u. The key partner set is defined as $KP(u) = \{u_i | \forall u_i \in U \setminus \{u\}(E_u \subseteq E_{u_i})\}$.*

Lemma 3. *Given a query user u_q, $\forall e \in E\ \forall u \in KP(u_q)\ \forall u' \in U \setminus (KP(u_q) \cup \{u_q\})\ (p(u_q, e, u) = 1 \geq p(u_q, e, u'))$.*

Proof. Since $\forall u \in KP(u_q)(E_u \supseteq E_{u_q})$ and $\forall e \in E(E_{u_q} \supseteq N(u_q, e))$, we have $\forall u \in KP(u_q)\forall e \in E(E_u \supseteq N(u_q, e))$ and derive that $p(u_q, e, u) = 1$. Straightforwardly, $\forall u' \in U \setminus (KP(u_q) \cup \{u_q\})(p(u_q, e, u) = 1 \geq p(u_q, e, u'))$.

The key partner set of a query user may contain multiple users. According to the definition of the kEP query, the events in the result are distinct. Thus, we arbitrarily select one user from the key partner set for each event.

4.3 Efficient Partner Computation

In the framework, 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 for each event that consists of pairs (u, num) sorted descendingly on num, which is the number of events attended by user u.

Table 5 shows the event-member lists of the five events in Fig. 1. For instance, event e_4 has members u_4 and u_5. User u_4 has attended five events, and user u_5 has attended three events.

Table 5. 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 1 shows the pseudo code of the efficient partner computation. It takes a query user u_q, an event e, and a neighborhood $N(u_q, e)$ as arguments, and it returns the partner user u who maximizes the together preference $p(u_q, e, u)$.

Given neighborhood $N(u_q, e)$, 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 $p(u_q, e, u)$ is larger than 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 4) on the together preference 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 1. ComputePartner$(N(u_q, e), u_q, e)$

```
 1: $p_1 \leftarrow -1$
 2: $p_r \leftarrow +\infty$
 3: $u_p \leftarrow null$
 4: Fetch the member list $ml(e_i)$ of each event in $N(u_q, e)$
 5: while $p_1 < p_r$ do                                  ▷ Pruning Rule 2
 6:     $(u, num) \leftarrow$ GetNextPair()
 7:     if $p_1 < p(u_q, e, u)$ then
 8:         $p_1 \leftarrow p(u_q, e, u)$
 9:         $u_p \leftarrow u$
10:     end if
11:     Remove $(u, num)$ from the member list
12:     $(u, num) \leftarrow$ GetNextPair()
13:     $x \leftarrow \min\{num, |N(u_q, e)|\}$
14:     $p_r \leftarrow (\sum_{e_i \in TopX(u_q,e)} s(e, e_i))/(\sum_{e_i \in N(u_q,e)} s(e, e_i))$
15: end while
16: return $u_p$
```

Lemma 4. *Given a query user u_q and an event e, let (u, num) be the pair returned by function* **GetNextPair**() *and define $x = \min\{num, |N(u_q, e)|\}$. Then set $TopX(u_q, e)$ contains the top-x events $\{e_i\}$ in neighborhood $N(u_q, e)$ 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 $N(u_q, e)$ is*

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

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 $N(u_q, e)$. This means that no user in the member lists of the events in $N(u_q, e)$ can have attended more events than the returned *num*. Since $x = \min\{num, |N(u_q, e)|\}$, no user in the member lists of the events in $N(u_q, e)$ has attended more than x events in $N(u_q, e)$. Given that set $TopX(u_q, e)$ contains the top-x events $\{e_i\}$ in neighborhood $N(u_q, e)$ with the largest similarity $s(e, e_i)$, then, for any user u_i in the member lists of the events in $N(u_q, e)$, we have $p_r \geq p(u_q, e, u_i)$.

Pruning Rule 2 *In the context of Algorithm 1, 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 are 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 5, the pair returned by function **GetNextPair**() is $(u_3, 4)$. The together preference $p(u_4, e_2, u_3) = 1$. 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 $p_r = 0.5/(0.5 + 0.3) = 0.63$. 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 Empirical Study

We proceed to cover a study of the performance of the proposed framework and its three optimizations. In the experiments, F-NLJ and F-NLJ* denote the framework adopting the nested loop join without and with optimizations, respectively. F-RJ* denotes the framework adopting the ripple join with optimizations.

5.1 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.

We generated 5 query sets, in which the number of keywords is 1, 2, 3, 4, and 5, respectively, taken from the data set. Each query set comprises 100 queries. Specifically, to generate a query, we randomly pick a user in the data set as the query user, and we randomly choose words from the document of a randomly selected event as the query keywords. We ensure that no query has an empty result.

[3] http://www.meetup.com.

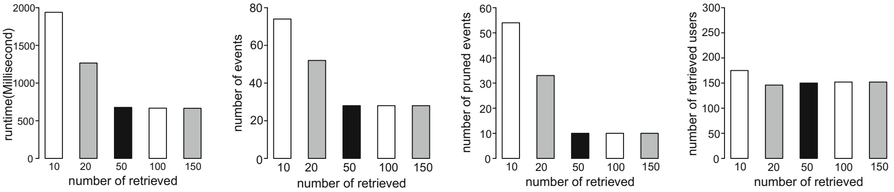
5.2 Setup

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 key partner sets of all users are kept in main memory. In the data set, 68% of all users have key partner sets. 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.

We study the effects of different parameters and set parameter default values 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 (Eq. 2) is 0.5. Some queries take long time to compute using the framework without optimizations. We set 20 s as a time limit. If the processing of any query exceeds 20 s, we stop the processing.

5.3 Performance Evaluation

Tuning the Number of Retrieved Events and Users in the Ripple Join. When using the ripple join, the numbers of retrieved events and users at a time affects the performance. We thus evaluate the framework using the ripple join when varying the numbers of retrieved events and users. It is observed that varying the number of retrieved events does not affect the performance, while varying the number of retrieved users does, as shown in Fig. 2. The runtime improves as the number of retrieved users is increased from 10 to 100, and it gets slightly worse when the number of retrieved users is increased from 100 to 150. The number of events and users involved in the computation and the number of pruned events exhibit similar behavior. Thus, in the following experiment, the number of retrieved users in the ripple join is set to 100.

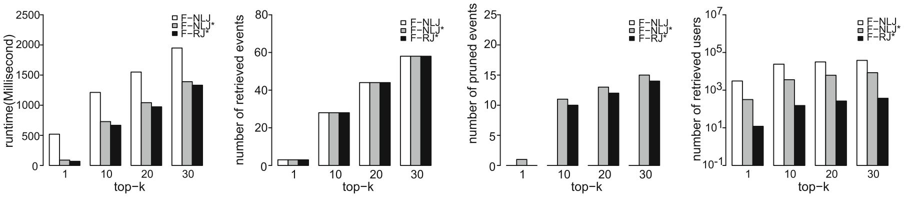


(a) Runtime (b) Events (c) Pruned Events (d) Users Per Event

Fig. 2. Varying the number of retrieved users in ripple join

[4] https://lucene.apache.org.

Varying the Number k of Requested Event-Partner Pairs. Figure 3 shows the average runtime, the average number of retrieved events per query, the average number of pruned events, and the average number of retrieved users per event when varying k. The average runtime of the three approaches increases as k increases, since more and more relevant events are retrieved and more and more users are involved as k increases. The unpromising event pruning optimization prunes many events, as shown in Fig. 3(c). The efficient partner computation optimization reduces the number of retrieved users per event, as shown in Fig. 3(d). F-NLJ has almost the same number of retrieved events as each of F-NLJ* and F-RJ* (Fig. 3(b)). This is because F-NLJ has significantly more retrieved users per event than do F-NLJ* and F-RJ*, so that the processing time of some of the queries exceeds the 20 s time limit, forcing those queries to stop. 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 also has significantly fewer retrieved users per event than does F-NLJ*, which means that the ripple join is better than the nested loop join. Hence, F-RJ* outperforms F-NLJ* in terms of runtime.

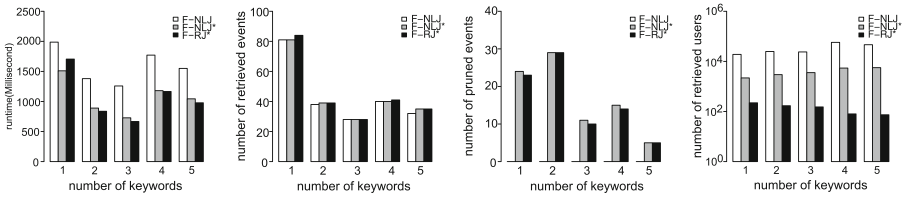


(a) Runtime (b) Events (c) Pruned Events (d) Users Per Event

Fig. 3. Varying the number of requested event-partner pairs k

Varying the Number of Query Keywords. Figure 4 shows the average runtime, the average number of retrieved events per query, the average number of pruned events, and the average number of retrieved users per event when varying the number of query keywords. It can also be seen that the unpromising event pruning and efficient partner computation optimizations are effective, cf. Figure 4(c) and (d). The number of events retrieved by F-NLJ is slightly lower than those of both F-NLJ* and F-RJ* (Fig. 4(b)). This occurs because the processing of some of the queries using F-NLJ take too long and are forced to stop. Overall, F-NLJ* and F-RJ* outperform F-NLJ significantly in terms of runtime.

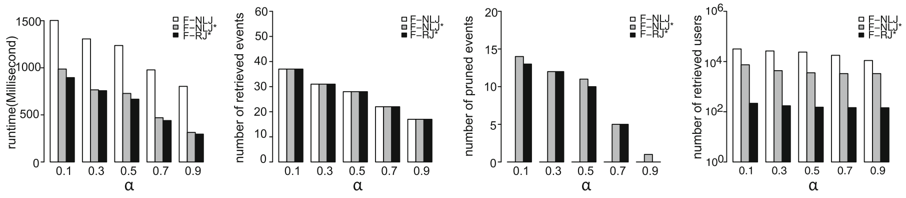
Varying α. Figure 5 reports on the finding when varying α that specifies the weight of the textual relevance in the scoring function. The performance of the three approaches get better (shorter runtime, fewer retrieved events and users) as α increases. The reason is that a large α gives high weight to the textual relevance, so that the ranking of the event-partner pairs is affected more by the textual relevance of the events than the together preference. Since events are retrieved in descending order of the textual relevance in the three approaches,



(a) Runtime (b) Events (c) Pruned Events (d) Users Per Event

Fig. 4. Varying the number of keywords

the top-k event-partner pairs are determined faster when α is large. Consistent with the previous results, this experiment also shows the effectiveness of the propose optimizations, i.e, many unpromising events are pruned, and the number of retrieved users per event is reduced.



(a) Runtime (b) Events (c) Pruned Events (d) Users Per Event

Fig. 5. Varying α

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 reduced. In most cases, the ripple join is better than the nested loop join.

6 Related Work

Recommender Systems. Friend recommendation systems [8] predict user-user relationships (i.e., friendships). They estimates the likelihood that two non-friends will become friends in the future [17,28,31]. Group recommendation [6, 32] explores the preference of a group of users in relation to individual items. In location-based social networks, new locations are recommended to users [29] by taking into account previous user check-ins, the distances of proposed locations to users' neighborhoods [1], and geographical and social information [16]. In event-based social networks, event recommendation offers a user a set of events by giving consideration to both personal interests and local preferences [30], heterogeneous social relations and implicit feedback [24], social group influences and individual preferences [5], and several contextual signals [18].

All these recommendation techniques only recommend one type of items. The problem studied in this paper adopts a recent recommendation technique [27] that suggests event-partner pairs as a component.

Rank-Join Algorithms. Based on the A* optimization strategy, the J^* algorithm [21] enables querying of ordered data sets by means of user-defined join predicates. NRA-RJ [9] is a pipelined query operator that produces a global rank from ranked input streams based on a scoring function. Ilyas et al. [10] rank join results progressively during join operations, making use of the individual orders of the inputs.

Ranking (top-k) queries have also been integrated into relational database systems [11,13]. Mamoulis et al. [19] identify two phases that any (no 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 [4] 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) [26] 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 [10] and propose a framework with optimizations that supports the efficiently processing of kEP queries.

7 Conclusion

This paper introduces the top-k event-partner (kEP) pair retrieval query that takes the advantages of both event-partner recommendation and keyword-based search. Given a query user, keywords, and a value k, the query retrieves event-partner pairs from a bipartite event-user graph where events have text descriptions, taking into account both the text relevance of events and the together preference that captures how much the query user prefers to attend an event with a particular partner. For the sake of efficiency, the proposed rank-join based 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.

This work opens to a number of promising directions for future work. First, it is worth adapting other existing recommendation techniques developed for events and users to the paper's setting. Second, it is of interest to consider social relationships between users when processing kEP queries. Third, it is of interest to understand how the kEP queries considered can be best processed if the query user's current location is taken into account.

References

1. Bao, J., Zheng, Y., Wilkie, D., Mokbel, M.F.: Recommendations in location-based social networks: a survey. GeoInformatica **19**(3), 525–565 (2015)
2. Chen, X., Zeng, Y., Cong, G., Qin, S., Xiang, Y., Dai, Y.: On information coverage for location category based point-of-interest recommendation. In: AAAI, pp. 37–43 (2015)
3. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. **66**(4), 614–656 (2003)
4. Finger, J., Polyzotis, N.: Robust and efficient algorithms for rank join evaluation. In: SIGMOD, pp. 415–428 (2009)
5. Gao, L., Wu, J., Qiao, Z., Zhou, C., Yang, H., Hu, Y.: Collaborative social group influence for event recommendation. In: CIKM, pp. 1941–1944 (2016)
6. Gorla, J., Lathia, N., Robertson, S., Wang, J.: Probabilistic group recommendation via information matching. In: WWW, pp. 495–504 (2013)
7. Haas, P.J., Hellerstein, J.M.: Ripple joins for online aggregation. In: SIGMOD, pp. 287–298 (1999)
8. Hannon, J., Bennett, M., Smyth, B.: Recommending twitter users to follow using content and collaborative filtering approaches. In: RecSys, pp. 199–206 (2010)
9. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Joining ranked inputs in practice. In: VLDB, pp. 950–961 (2002)
10. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting top-k join queries in relational databases. VLDB J. **13**(3), 207–221 (2004)
11. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K., Elmongui, H.G., Shah, R., Vitter, J.S.: Adaptive rank-aware query optimization in relational databases. ACM Trans. Database Syst. **31**(4), 1257–1304 (2006)
12. Ji, X., Qiao, Z., Xu, M., Zhang, P., Zhou, C., Guo, L.: Online event recommendation for event-based social networks. In: WWW, pp. 45–46 (2015)
13. Li, C., Chang, K.C., Ilyas, I.F., Song, S.: RankSQL: query algebra and optimization for relational top-k queries. In: SIGMOD, pp. 131–142 (2005)
14. Li, H., Ge, Y., Hong, R., Zhu, H.: Point-of-interest recommendations: learning potential check-ins from friends. In: KDD, pp. 975–984 (2016)
15. Lian, D., Zhao, C., Xie, X., Sun, G., Chen, E., Rui, Y.: GeoMF: joint geographical modeling and matrix factorization for point-of-interest recommendation. In: KDD, pp. 831–840 (2014)
16. Liu, B., Fu, Y., Yao, Z., Xiong, H.: Learning geographical preferences for point-of-interest recommendation. In: KDD, pp. 1043–1051 (2013)
17. Lu, Y., Qiao, Z., Zhou, C., Hu, Y., Guo, L.: Location-aware friend recommendation in event-based social networks: a Bayesian latent factor approach. In: CIKM, pp. 1957–1960 (2016)
18. Macedo, A.Q., Marinho, L.B., Santos, R.L.: Context-aware event recommendation in event-based social networks. In: RecSys, pp. 123–130 (2015)
19. Mamoulis, N., Yiu, M.L., Cheng, K.H., Cheung, D.W.: Efficient top-k aggregation of ranked inputs. ACM Trans. Database Syst. **32**(3), 19 (2007)
20. Manotumruksa, J., MacDonald, C., Ounis, I.: Regularising factorised models for venue recommendation using friends and their comments. In: CIKM, pp. 1981–1984 (2016)
21. Natsev, A., Chang, Y., Smith, J.R., Li, C., Vitter, J.S.: Supporting incremental join queries on ranked inputs. In: VLDB, pp. 281–290 (2001)

22. Ponte, J.M., Croft, W.B.: A language modeling approach to information retrieval. In: SIGIR, pp. 275–281 (1998)
23. Qiao, Z., Zhang, P., Cao, Y., Zhou, C., Guo, L., Fang, B.: Combining heterogenous social and geographical information for event recommendation. In: AAAI, pp. 145–151 (2014)
24. Qiao, Z., Zhang, P., Zhou, C., Cao, Y., Guo, L., Zhang, Y.: Event recommendation in event-based social networks. In: AAAI, pp. 3130–3131 (2014)
25. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Commun. ACM **18**(11), 613–620 (1975)
26. Schnaitter, K., Polyzotis, N.: Optimal algorithms for evaluating rank joins in database systems. ACM Trans. Database Syst. **35**(1), 6:1–6:47 (2010)
27. Tu, W., Cheung, D.W., Mamoulis, N., Yang, M., Lu, Z.: Activity-partner recommendation. In: Cao, T., Lim, E.-P., Zhou, Z.-H., Ho, T.-B., Cheung, D., Motoda, H. (eds.) PAKDD 2015. LNCS (LNAI), vol. 9077, pp. 591–604. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18038-0_46
28. Wan, S., Lan, Y., Guo, J., Fan, C., Cheng, X.: Informational friend recommendation in social media. In: SIGIR, pp. 1045–1048 (2013)
29. Wang, W., Yin, H., Sadiq, S.W., Chen, L., Xie, M., Zhou, X.: SPORE: a sequential personalized spatial item recommender system. In: ICDE, pp. 954–965 (2016)
30. Yin, H., Sun, Y., Cui, B., Hu, Z., Chen, L.: LCARS: a location-content-aware recommender system. In: KDD, pp. 221–229 (2013)
31. Yu, F., Che, N., Li, Z., Li, K., Jiang, S.: Friend recommendation considering preference coverage in location-based social networks. In: Kim, J., Shim, K., Cao, L., Lee, J.-G., Lin, X., Moon, Y.-S. (eds.) PAKDD 2017. LNCS (LNAI), vol. 10235, pp. 91–105. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57529-2_8
32. Yuan, Q., Cong, G., Lin, C.: COM: a generative model for group recommendation. In: KDD, pp. 163–172 (2014)
33. Zhang, W., Wang, J.: A collective Bayesian Poisson factorization model for cold-start local event recommendation. In: KDD, pp. 1455–1464 (2015)