A Multiple Layer Efficient and Scalable Location Aware Recommender System

Mrs.R.Bhavani, AP/CSE, Ramya.K
M.Tech(Computer Science),PRIST University,Thanjavur.

ABSTRACT

This paper proposes LARS*, a location-aware recommender system that uses location-based ratings to produce recommendations. Traditional recommender systems do not consider spatial properties of users nor items; LARS*, on the other hand, supports a taxonomy of three novel classes of location-based ratings, namely, spatial ratings for non-spatial items, non-spatial ratings for spatial items, and spatial ratings for spatial items. LARS* exploits user rating locations through user partitioning, a technique that influences recommendations with ratings spatially close to querying users in a manner that maximizes system scalability while not sacrificing recommendation quality. LARS* exploits item locations using travel penalty, a technique that favors recommendation candidates closer in travel distance to querying users in a way that avoids exhaustive access to all spatial items. LARS* can apply these techniques separately, or together, depending on the type of location-based rating available. Experimental evidence using large-scale real-world data from both the Foursquare location-based social network and the MovieLens movie recommendation system reveals that LARS* is efficient, scalable, and capable of producing recommendations twice as accurate compared to existing recommendation approaches.

Key Terms—Recommender system, spatial, location, performance, efficiency, scalability, ratings

1. INTRODUCTION

Recommender systems make use of community opinions to help users identify useful items from a considerably large search space. A location-content-aware recommender system that offers a particular user a set of venues (e.g., restaurants) or events (e.g., concerts and exhibitions) by giving consideration to both personal interest and local preference. This recommender system can facilitate people’s travel not only near the area in which they live, but also in a city that is new to them. A location-content-aware recommender system that offers a particular user a set of venues (e.g., restaurants) or events (e.g., concerts and exhibitions) by giving consideration to both personal interest and local preference. This recommender system can facilitate people’s travel not only near the area in which they live, but also in a city that is new to them.

Location Aware Recommender System provides an efficient location-based search on network, which leads a high confliction while travelling around in a world. But this project provides high-quality location-based recommendations and served as best in an efficient manner. This project includes three types of location-based ratings within single framework.

1.1 Spatial Ratings for Non-Spatial Items

It includes four attributes they are, user, ulocation, rating, item, in this ulocation represents a user location denote where the user is currently located. For example, a user located at home rating a book.

1.2 Non-Spatial Ratings for Spatial Items

Similar as first type this also includes four attributes namely user, rating, item, ilocation, where ilocation represents an item location in this item represents some unmovable things which are provide some landmark to this system. For example, a user with unknown location rating a restaurant.

1.3 Spatial Ratings for Spatial Items

This type contains five attributes they are user, ulocation, rating, item, ilocation. It contains two techniques a user partitioning technique that exploits user locations in that way it increases the system scalability as well as it doesn’t sacrifice the recommendation locality and a travel penalty technique that
exploits item locations and avoids exhaustively processing all spatial recommendation candidates. Location aware recommendersystem achieves higher locality gain than previous location aware systems using a better user partitioning data structure and algorithm. It also exhibits a more flexible tradeoff between locality and scalability for large systems. Alongwith Location Aware recommender system also provides a more efficient way to maintain the user partitioning structure

2. RELATED WORK

2.1 LCARS: A Location-Content-Aware Recommender System

LCARS, a location-content-aware recommendersystem that offers a particular user a set of venues (e.g., restaurants) or events (e.g., concerts and exhibitions) by giving consideration to both personal interest and local preference. This recommender system can facilitate people’s travel not only near therea in which they live, but also in a city that is new to them. Specifically, LCARS consists of two components: offline modeling and online recommendation. The offline modeling part, called LCALDA, is designed to learn the interest of each individual user and the local preference of each individual city by capturing item co-occurrence patterns and exploiting item contents. The online recommendation part automatically combines the learnt interest of the querying user and the local preference of the querying city to produce the top-k recommendations. To speed up this online process, a scalable query processing technique is developed by extending the classic Threshold Algorithm. We evaluate the performance of four recommender system on two large-scale real data sets, Douban-Event and Foursquare. The results show the superiority of LCARSin recommending spatial items for users, especially when traveling to new cities, in terms of both effectiveness and efficiency.

ADVANTAGES

- Provide high performance for large datasets
- System efficiency was high.

DISADVANTAGES

- Lack of knowledge for retrieving content-based queries.

2.2 Location Recommendation in Location-Based Social Networks Using User Check-In Data

As an increasingly larger number of users partake in LBSNs(Location based social network), the recommendation problem in this setting has attracted significant attention in research and in practical applications. The detailed information about past user behavior that is traced by the LBSN differentiates the problem significantly from its traditional settings. The spatial nature in the past user behavior and also the information about the user social interaction with other users, provide a richer background to build a more accurate and expressiverecommendation model.

ADVANTAGES

- Fast retrieval of data
- Performance and Efficiency of the system will increase.

DISADVANTAGES

- Data cannot be secured.
- Sometimes provide un relevant information

2.3 Efficient Evaluation of k-Range Nearest Neighbor Queries in Road Networks

The kRNN query is significantly important for location-based applications in many realistic scenarios. For example, (1) the user’s location is uncertain, i.e., user’s location is modeled by a spatial region, and (2) the user is not willing to reveal her exact location to preserve her privacy, i.e., her location is blurred into a spatial region. However, the existing solutions for kRNN queries simply apply the traditional k-nearest neighbor query processing algorithm multiple times, which poses a huge redundant searching overhead. To this end, we propose an efficient kRNN query processing algorithm in this paper. Our algorithm (1) employs a shared execution approach to eliminate the redundant searching overhead, and (2) provides a parameter that can be tuned to achieve a tradeoff between the
query processing performance and the storage overhead, while guaranteeing the user’s exact k-nearest neighbors are included in the query answers.

**ADVANTAGES**

- Eliminates redundant information
- Avoids overhead

**DISADVANTAGES**

- Requires a large database.

---

### 2.4 LARS* Query Model

Users (or applications) provide LARS* with a user id $U$, numeric limit $K$, and location $L$; LARS* then returns $K$ recommended items to the user. LARS* supports both snapshot (i.e., one-time) queries and continuous queries, whereby a user subscribes to LARS* and receives recommendation updates as her location changes. The technique LARS* uses to produce recommendations depends on the type of location-based rating available in the system. Query processing support for each type of location-based rating is discussed in Sections 4 to 6.

---

### 2.2 Item-Based Collaborative Filtering

LARS* uses item-based collaborative filtering (abbr. CF) as its primary recommendation technique, chosen due to its popularity and widespread adoption in commercial systems (e.g., Amazon [1]). Collaborative filtering (CF) assumes a set of $n$ users $U = \{u_1, \ldots, u_n\}$ and a set of $m$ items $I = \{i_1, \ldots, i_m\}$. Each user expresses opinions about a set of items $I_{uj} \subseteq I$. Opinions can be a numeric rating (e.g., the Netflix scale of one to five stars), or unary (e.g., Facebook “check-ins” [3]). Conceptually, ratings are represented as a matrix with users and items as dimensions, as depicted in Fig. 2(a). Given a querying user $u$, CF produces a set of $k$ recommended items $Ir \subseteq I$ that $u$ is predicted to like the most.

**Phase I: Model Building.** This phase computes a similarity score $sim(ip, iq)$ for each pair of objects $ip$ and $iq$ that have at least one common rating by the same user (i.e., co-rated dimensions). Similarity computation is covered below. Using these scores, a model is built that stores for each item $i \in I$, a list $L$ of similar items ordered by a similarity score $sim(ip, iq)$, as depicted in Fig. 2(b). Building this model is an $O(R2)$ process [1], where $R$ and $U$ are the number of ratings and users, respectively. It is common to truncate the model by storing, for each list $L$, only the $n$ most similar items with the highest similarity scores [9]. The value of $n$ is referred to as the model size and is usually much less than $|I|$.

**Phase II: Recommendation Generation.** Given a querying user $u$, recommendations are produced by computing $u$’s predicted rating $P(u,i)$ for each item $i$ not rated by $u$ [9]:

$$P(u,i) = \sum_{l \in L} L_{\text{sim}(i, l) \neq 0} \sum_{u,l \in L} L_{\text{sim}(i, l) \neq 0}$$

(1)

Before this computation, we reduce each similarity list $L$ to contain only items rated by user $u$. The prediction is the sum of $ru,l$, a user $u$’s rating for a related item $l \in L$ weighted by $\text{sim}(i,l)$, the similarity of $i$ to candidate item $i$, then normalized by the sum of similarity scores between $i$ and $l$. The user receives as recommendations the top-$k$ items ranked by $P(u,i)$.

**Computing Similarity.** To compute $sim(ip, iq)$, we represent each item as a vector in the user-rating space of the rating matrix. For instance, Fig. 3 depicts vectors for items $ip$ and $iq$ from the matrix in Fig. 2(a). Many similarity functions have been proposed (e.g., Pearson Correlation, Cosine); we use the Cosine similarity in LARS* due to its popularity:

$$sim(ip, iq) = _ip \cdot _iq - _ip \cdot _iq - _ip \cdot _iq$$

(2)

This score is calculated using the vectors’ co-rated dimensions, e.g., Cosine distance is useful for numeric ratings (e.g., on a scale $[1,5]$). For unary ratings, other similarity functions are used (e.g., absolute sum $10]$). While we opt to use item-based CF in this paper, no factors disqualify us from employing other recommendation techniques. For instance, we could easily employ user-based CF [4], that uses correlations between users (instead of items).
3 NON-SPATIAL USER RATINGS FOR NON-SPATIAL ITEMS
The traditional item-based collaborative filtering (CF) method is a special case of LARS*. CF takes as input the classical rating triplet (user, rating, item) such that neither the user location nor the item location are specified. In such case, LARS* directly employs the traditional model building phase (Phase-I in section 2) to calculate the similarity scores between all items. Moreover, recommendations are produced to the users using the recommendation generation phase (Phase-II in section 2). During the rest of the paper, we explain how LARS* incorporates either the user spatial location or the item spatial location to serve location-aware recommendations to the system users.

4 SPATIAL USER RATINGS FOR NON-SPATIAL ITEMS
This section describes how LARS* produces recommendations using spatial ratings for non-spatial items represented by the tuple (user, location, rating, item). The idea is to exploit preference locality, i.e., the observation that user opinions are spatially unique. We identify three requirements for producing recommendations using spatial ratings for non-spatial items: (1) Locality: recommendations should be influenced by those ratings with user locations spatially close to the querying user location (i.e., in a spatial neighborhood); (2) Scalability: The recommendation procedure and data structure should scale up to large number of users; (3) Influence: system users should have the ability to control the size of the spatial neighborhood (e.g., city block, zip code, or county) that influences their recommendations. LARS* achieves its requirements by employing a user-partitioning technique that maintains an adaptive pyramindature, where the shape of the adaptive pyramid is driven by the three goals of locality, scalability, and influence. The idea is to adaptively partition the rating tuples (user, location, rating, item) into spatial regions based on the location attribute. Then, LARS* produces recommendations using any existing collaborative filtering method (we use item-based CF) over the remaining three attributes (user, rating, item) of only the ratings within the spatial region containing the querying user. We note that ratings can come from users with varying tastes, and that our method only forces collaborative filtering to produce personalized user recommendations based only on ratings restricted to a specific spatial region.

4.1 Data Structure Maintenance
We note that while the original partial pyramid [11] was concerned with
Algorithm 1
Pyramid maintenance algorithm
1: /* Called after cell C receives N% new ratings */
2: Function PyramidMaintenance(Cell C, Level h)
3: /* Step I: Statistics Maintenance*/
4: Maintain cell C statistics
5: /*Step II: Model Rebuild */
6: if (Cell C is an α-Cell) then
7: Rebuild item-based collaborative filtering model for cell C
8: end if
9: /*Step III: Cell Child Quadrant Maintenance*/
10: if (C children quadrant q cells are α-Cells) then
11: CheckDownGradeToSCells(q,C) /* covered in Section 4.5.2 */
12: else if (C children quadrant q cells are γ-Cells) then
13: CheckUpGradeToSCells(q,C)
14: else
15: isSwitchedToMcells← CheckUpGradeToMCells(q,C) /* covered in Section 4.5.3 */
16: if (isSwitchedToMcells is False) then
17: CheckDownGradeToECells(q,C)
18: end if
19: end if
20: return
spatial queries over static data, it did not address pyramid maintenance.
4.2 Main Idea
As time goes by, new users, ratings, and items will be added to the system. This new data will both increase the size of the collaborative filtering models maintained in the pyramids, as well as alter recommendations produced from
each cell. To account for these changes, LARS* performs maintenance on a cell-by-cell basis. Maintenance is triggered for a cell \( C \) once it receives \( N\% \) new ratings; the percentage is computed from the number of existing ratings in \( C \). We do this because an appealing quality of collaborative filtering is that as a model matures (i.e., more data is used to build the model), more updates are needed to significantly change the top-k recommendations produced from it \([17]\). Thus, maintenance is needed less often.

We note the following features of pyramid maintenance:

1. Maintenance can be performed completely offline, i.e., LARS* can continue to produce recommendations using the "old" pyramid cells while part of the pyramid is being updated;
2. Maintenance does not entail rebuilding the whole pyramid at once, instead, only one cell is rebuilt at a time; and
3. Maintenance is performed only after \( N\% \) new ratings are added to a pyramid cell, meaning maintenance will be amortized over many operations.

### 4.3 Maintenance Algorithm

Algorithm 1 provides the pseudocode for the LARS* maintenance algorithm. The algorithm takes as input a pyramid cell \( C \) and level \( h \), and includes three main steps:

- **Statistics Maintenance**,
- **Model Rebuild**,
- **Cell Child Quadrant Maintenance**, explained below.

#### Step I: Statistics Maintenance.**

The first step (line 4) is to maintain the Items Ratings Statistics Table. The maintained statistics are necessary for cell type switching decision, especially when new location-based ratings enter the system. As the items ratings statistics table is implemented using a hash table, then it can be queried and maintained in \( O(1) \) time, requiring \( O(|IC|) \) space such that \( IC \) is the set of all items rated at cell \( C \) and \( |IC| \) is the total number of items in \( IC \).

#### Step II: Model Rebuild.

The second step is to rebuild the item-based collaborative filtering (CF) model for a cell \( C \), as described in Section 2.2 (line 7). The model is rebuilt at cell \( C \) only if cell \( C \) is an α-Cell, otherwise (β-Cell or γ-Cell) no CF recommendation model is maintained, and hence the model rebuild step does not apply. Rebuilding the CF model is necessary to allow the model to "evolve" as new location-based ratings enter the system (e.g., accounting for new items, ratings, or users). Given the cost of building the item-based CF model is \( O(R2U) \) (per Section 2.2), the cost of the model rebuild for a cell \( C \) at level \( h \) is \( (R/4h)(U/4h) = R2hU \), assuming ratings and users are uniformly distributed.

### Step III: Cell Child Quadrant Maintenance.

LARS* invokes a maintenance step that may decide whether cell \( C \) child quadrant need to be switched to a different cell type based on trade-offs between scalability and locality. The algorithm first checks if cell \( C \) child quadrant \( q \) at level \( h + 1 \) is of type \( α \)-Cell (line 10). If that case holds, LARS* considers quadrant \( q \) cells as candidates to be downgraded to β-Cells (calling function \( \text{CheckDownGradeToCell} \) on line 11). We provide details of the Downgrade α-Cells to β-Cells operation in Section 4.5.2. On the other hand, if \( C \) have a child quadrant of type \( γ \)-Cells at level \( h + 1 \) (line 12), LARS* considers upgrading cell \( C \) four children cells at level \( h + 1 \) to β-Cells (calling function \( \text{CheckUpGradeToCell} \) line 13). The Upgrade to β-Cells operation is covered in Section 4.5.4. However, if \( C \) has a child quadrant of type β-Cells at level \( h + 1 \) (line 12), LARS* first considers upgrading cell \( C \) four children cells at level \( h + 1 \) from β-Cells to α-Cells (calling function \( \text{CheckUpGradeToCell} \) line 15). If the children cells are not switched to α-Cells, LARS* then considers downgrading them to γ-Cells (calling function \( \text{CheckDownGradeToCell} \) line 17). Cell Type switching operations are performed completely in quadrants (i.e., four quadrant cells with the same parent). We made this decision for simplicity in maintaining the partial pyramid.
5 NON-SPATIAL USER RATINGS FOR SPATIAL ITEMS

This section describes how LARS* produces recommendations using non-spatial ratings for spatial items represented by the tuple (user, rating, item, location). The idea is to exploit travel locality, i.e., the observation that users limit their choice of spatial venues based on travel distance (based on analysis in Section 1.1). Traditional (non-spatial) recommendation techniques may produce recommendations with burdensome travel distances (e.g., hundreds of miles away). LARS* produces recommendations within reasonable travel distances by using travel penalty, a technique that penalizes the recommendation rank of items the further in travel distance they are from a querying user. Travel penalty may incur expensive computational overhead by calculating travel distance to each item. Thus, LARS* employs an efficient query processing technique capable of early termination to produce the recommendations without calculating the travel distance to all items. Section 5.1 describes the query processing framework while Section 5.2 describes travel distance computation.

5.1 Query Processing

Query processing for spatial items using the travel penalty technique employs a single system-wide item-based collaborative filtering model to generate the top-k recommendations by ranking each spatial item i for a querying user u based on $\text{RecScore}(u, i)$, computed as:

$$\text{RecScore}(u, i) = \text{P}(u, i) - \text{TravelPenalty}(u, i).$$

(7)

P(u, i) is the standard item-based CF predicted rating of item i for user u (see Section 2.2). TravelPenalty(u, i) is the road network travel distance between u and i normalized to the same value range as the rating scale (e.g., [0, 5]). When processing recommendations, we aim to avoid calculating Equation 7 for all candidate items to find the top-k recommendations, which can become quite expensive given the need to compute travel distances. To avoid such computation, we evaluate items in monotonically increasing order of travel penalty (i.e., travel distance), enabling us to use early termination principles from top-k query processing[18]–[20]. We now present the main idea of our query processing algorithm and in the next section discuss how to compute travel penalties in an increasing order of travel distance. Algorithm 2 provides the pseudo code of our query processing algorithm that takes a querying user id U, a

Algorithm 2 Travel Penalty Algorithm for Spatial Items

1: Function LARS*_SpatialItems(User U, Location L, Limit K)
2: /* Populate a list R with a set of K items*/
3: R $\leftarrow$ $\phi$
4: for (K iterations) do
5: i $\leftarrow$ Retrieve the item with the next lowest travel penalty (Section 5.2)
6: Insert i into R ordered by RecScore(U, i) computed by Equation 7
7: end for
8: LowestRecScore $\leftarrow$ RecScore of the kth object in R
9: /* Retrieve items one by one in order of their penalty value */
10: while there are more items to process do
11: i $\leftarrow$ Retrieve the next item in order of penalty score (Section 5.2)
12: MaxPossibleScore $\leftarrow$ MAX_RATING - i.penalty
13: if MaxPossibleScore $\leq$ LowestRecScore then
14: return R /* early termination - end query processing */
15: end if
16: RecScore(U, i) $\leftarrow$ P(U, i) - i.penalty/* Equation 7 */
17: if RecScore(U, i) > LowestRecScore then
18: Insert i into R ordered by RecScore(U, i)
19: LowestRecScore $\leftarrow$ RecScore of the kth object in R
20: end if
21: end while
22: return R

location L, and a limit K as input, and returns the list R of top-k recommended items. The algorithm starts by running an $k$-nearest-neighbor algorithm to populate the list R with items with lowest travel penalty; R is sorted by the recommendationscore computed using Equation 7. This initial part is concluded by setting the lowest recommendation score value (LowestRecScore) as the RecScore of the kth item in R (Lines 3 to 8). Then, the algorithm starts to retrieve
items one by one in the order of their penalty score. This can be done using an incremental k-nearest-neighbour algorithm as will be described in the next section. For each item, we calculate the maximum possible recommendation score that i can have by subtracting the travel penalty of i from MAX_RATING, the maximum possible rating value in the system, e.g., 5 (Line 12). If i cannot make it into the list of top-k recommended items with this maximum possible score, we immediately terminate the algorithm by returning R as the top-k recommendations without computing the recommendation score (and travel distance) for more items (Lines 13 to 15). The rationale here is that since we are retrieving items in increasing order of their penalty and calculating the maximum score that any remaining item can have, then there is no chance that any unprocessed item can beat the lowest recommendation score in R.

If the early termination case does not arise, we continue to compute the score for each item i using Equation 7, insert i into R sorted by its score (removing the kth item if necessary), and adjust the lowest recommendation value accordingly (Lines 16 to 20). Travel penalty requires very little maintenance. The only maintenance necessary is to occasionally rebuild the system-wide item-based collaborative filtering model in order to account for new location-based ratings that enter the system. Following the reasoning discussed in Section 4.3, we rebuild the model after receiving N% new ratings.

6 SPATIAL USER RATINGS FOR SPATIAL ITEMS
This section describes how LARS* produces recommendations using spatial ratings for spatial items represented by the tuple (user, location, rating, item, iolocation). A salient feature of LARS* is that both the user partitioning and travel penalty techniques can be used together with very little change to produce recommendations using spatial user ratings for spatial items. The data structures and maintenance techniques remain exactly the same as discussed in Sections 4 and 5; only the query processing framework requires a slight modification. Query processing uses Algorithm 2 to produce recommendations. However, the only difference is that the item-based collaborative filtering prediction score P(u, i) used in the recommendation score calculation (Line 16 in Algorithm 2) is generated using the (localized) collaborative filtering model from the partial pyramid cell that contains the querying user, instead of the system-wide collaborative filtering model as was used in Section 5.

7. CONCLUSION
Location-aware recommender systems tackle a problem untouched by traditional recommender systems by dealing with three types of location-based ratings: spatial ratings for non-spatial items, non-spatial ratings for spatial items, and spatial ratings for spatial items. It employs user partitioning and travel penalty techniques to support spatial ratings and spatial items, respectively. Both techniques can be applied separately or in concert to support the various types of location-based ratings. Experimental analysis using real and synthetic datasets show that location aware recommender system is efficient, scalable, and provides better quality recommendations than techniques used in traditional recommender systems.

Location aware recommender systems exploits item locations using travel penalty, a technique that favors recommendation candidates closer in travel distance to querying users in a way that avoids exhaustive access to all spatial items. LARS can apply these techniques, or together, depending on the type of location-based rating available. LARS is efficient, scalable, and capable of producing recommendations twice as accurate compared to existing recommendation approaches. LARS addresses three possible types of location-based ratings. More importantly, LARS is a complete system (not just a recommendation technique) that employs efficiency and scalability techniques (e.g., merging, splitting, early query termination) necessary for deployment in actual large-scale applications. LARS evaluates a continuous query in full once it is issued, and sends recommendations back to a user as an initial answer. LARS then monitors the movement of user using her location updates. LARS does nothing as the initial answer is still
valid. In future, LARS can be made to answer more flexibly compare than now it has do. It can possibly ready to work even if it is answer cannot send to the user (i.e., that the user out of the boundary that the LARS can identify.

REFERENCES


