An Efficient Greedy Algorithm for Sequence Recommendation

Idir Benouaret, Sihem Amer-yahia, Senjuti Basu Roy

An Efficient Greedy Algorithm for Sequence Recommendation

Idir Benouaret[1], Sihem Amer-Yahia[1], and Senjuti Basu Roy[2]

[1] CNRS, Univ. Grenoble Alpes, France
[2] New Jersey Institute of Technology, Newark, NJ, USA
{idir.benouaret, sihem.amer-yahia}@univ-grenoble-alpes.fr
senjuti.basuroy@njit.edu

Abstract. Recommending a sequence of items that maximizes some objective function arises in many real-world applications. In this paper, we consider a utility function over sequences of items where sequential dependencies between items are modeled using a directed graph. We propose EdGe, an efficient greedy algorithm for this problem and we demonstrate its effectiveness on both synthetic and real datasets. We show that EdGe achieves comparable recommendation precision to the state-of-the-art related work OMEGA, and in considerably less time. This work opens several new directions that we discuss at the end of the paper.

Keywords: Sequence Recommendation · Submodular Maximization · Algorithms.

1 Introduction

Recommender Systems are models and algorithms that provide suggestions for items that are most likely to be of interest to a particular user [11]. Traditional recommendation systems are usually classified in two main categories. Collaborative Filtering approaches [13], which consists of learning users' preferences according to similar users and Content-based approaches, which consists of matching users' preferences with item features [8]. In their simplest form, recommender systems predict the most suitable items based on the user's preferences, and often return them as a ranked list. However, there are many applications where the order in which items should be consumed, and hence recommended, plays an important role in the satisfaction of the user. In that case, the set of recommended items should not be considered as a set of alternatives. Rather, they should form a sequence of items to be consumed in a "good" order. Typical examples include the recommendation of a sequence of music tracks [3], learning courses [7, 18], or points of interest (POIs) in a city [16].

While various existing efforts address the problem of recommending packages of items, e.g., [1, 17], they do not account for sequential dependencies and ordered preferences that might exist between items. In this paper, we are interested in developing a framework to recommend to a user u a sequence of at most k items

that maximizes a given utility function f. The function f should capture both the utility of selecting k items individually, and the utility of providing them in a given order $\sigma_1 \rightarrow \sigma_2, ..., \rightarrow \sigma_k$. For instance, a sequence containing a museum visit followed by sauna visit could have a higher utility than the inverse sequence. When dealing with sequences of items, the search space becomes exponential in the length of the recommended sequences. Submodular set functions constitute a category of widely used utility functions in recommender systems [5]. Such functions have been used to solve a submodular optimization problem that finds a subset of k items whose aggregated utility is maximized. The greedy algorithm that selects at each step the next item which maximizes a submodular monotone function has been shown to enjoy a $(1 - e^{-1})$- approximation guarantee [6]. Recently, Tschiatschek *et al.* [15] considered the sequence selection problem using an expressive class of utility functions over sequences, which generalizes the class of submodular functions and captures ordered preferences among items. They encoded these ordered preferences in a directed acyclic graph (DAG) over items, where a directed edge between two items models that, in addition to selecting the two items, there is an additional utility when selecting the "tail" item before selecting the "head" item that preserves their order. To the best of our knowledge, [15] is the only work that considers this submodular sequence recommendation problem, we will thus follow their problem formulation. The authors proposed OMEGA, an algorithm that exploits the DAG property of the underlying utility graph. For each set of items, its ordering can be computed by first finding a topological ordering of the graph and then sorting the set of items according to that. At each subsequent step, an edge is selected to maximize the given utility function according to the sequence of items induced by the selected edge at each step. However, the full sorting at each step coupled with computing the topological ordering of the graph leads to a high computational runtime. For example, selecting a sequence of 10 over 1000 items takes about 4 minutes in our settings.

In this paper, we propose EdGe, an Edge-based Greedy sequence recommendation algorithm that extends the classical greedy algorithm proposed in [6], but instead of selecting at each step the item (node) with a maximum gain, it selects the next valid edge with a maximum *gain according to the edges that are selected so far*. EdGe takes as input a directed graph of items and a submodular monotone function, and returns a sequence of length at most k. EdGe greatly improves the complexity of OMEGA [15]. The worst-case computational complexity of EdGe is $\mathcal{O}(m \cdot k)$, where m is the number of edges in the graph and k is the length of the recommended sequence.

Our experiments on both synthetic and real datasets verify the performance of EdGe in terms of response time and recommendation precision.

2 Problem Formulation

We now formally define the problem of recommending a sequence of items following the formulation in [15]. Let $\mathcal{V} = \{v_1, v_2, ..., v_n\}$ be the set of n items that

are available for recommendation. Our goal is to provide a user u with a sequence recommendation of length at most k, from the set of possible items, where the order in which items are recommended is important. Let $\Sigma = \{(\sigma_1, \sigma_2, ..., \sigma_k) | k \leq n, \sigma_1, \sigma_2, ..., \sigma_k \in \mathcal{V}, \forall i, j \in [k] : i \neq j \Rightarrow \sigma_i \neq \sigma_j\}$ be the set of all possible sequences of items without repetitions, that can be selected from the set of items $\mathcal{V}$. We denote by $|\sigma|$ the length of the sequence $\sigma \in \Sigma$, and for two sequences $\sigma, \pi \in \Sigma$, let $\sigma \oplus \pi$ be their concatenation.

Ordered preferences are modeled using a set of edges $\mathcal{E}$, which represents that there is a utility when selecting items in a certain order, as it is illustrated in Figure 1.

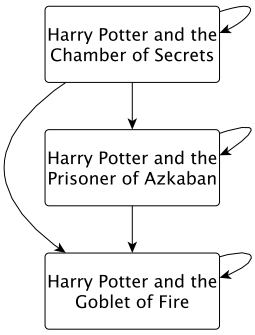

Fig. 1. An example of sequential dependencies between three *Harry potter* movies represented by a directed graph. Intuitively the order of watching these movies affects the satisfaction of the user. For example, watching *Harry Potter and the Chamber of Secrets* before watching *Harry Potter and the Prisoner of Azkaban* has a higher utility than watching them in the opposite order. Self-loops represent the utility of recommending an item individually.

More formally, an edge e_{ij} models the utility of selecting item v_j after item v_i. A self-loop edge e_{ii} models the utility of selecting item v_i individually. Hence, our model takes as input a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose nodes correspond to the set of items and whose edges correspond to the utilities of selecting items in a certain order. We define the utility function f for a sequence σ:

$$f(\sigma) = h(edges(\sigma)) \tag{1}$$

where $h(.)$ is a set-valued function and the function $edges(\sigma)$ maps the sequence of items σ to a set of induced edges according to the graph $\mathcal{G}$. More formally $edges(\sigma) = \cup_{j \in [|\sigma|]} \{(\sigma_i, \sigma_j) \mid (\sigma_i, \sigma_j) \in \mathcal{E},\ i \leq j\}$.

We assume that $h(E)$ is a non-negative monotone submodular set function over the edges $\mathcal{E}$, i.e, $h : 2^{\mathcal{E}} \rightarrow \mathbb{R}^{+*}$. Intuitively, submodularity describes the set of functions that satisfy a diminishing returns property. This property guarantees that the *marginal gain* of an edge $e \in \mathcal{E}$ is greater in the context of some set of

edges A compared to a larger set of edges $B \supseteq A$. formally $h(A \cup \{e\} - h(A)) \geq h(B \cup \{e\}) - h(B)$ for all sets A, B s.t. $A \subseteq B \subseteq \mathcal{E} \backslash \{e\}$. Particularly, in our experiments (Section 4), we use a probabilistic coverage function. Intuitively, the utility of a set of edges E is large if the nodes $nodes(E)$ are well covered.

$$h(E) = \sum_{j \in nodes(E)} [1 - \prod_{(i,j) \in E} (1 - w_{i,j})] \tag{2}$$

where $w_{i,j}$ is the utility associated with edge (i, j) and $nodes(E)$ is the set of items that are induced by the set of edges E.

Our recommendation task is to select a sequence σ of at most k unique items that will maximize f. This can be formalized as the following maximization problem:

$$max_{\sigma \in \Sigma, |\sigma| \leq k} f(\sigma) \tag{3}$$

The search space for optimal solutions is exponentially larger for sequences in terms of k. In particular, there are $\binom{|\mathcal{V}|}{k}$ subsets of items of size k but $k!\binom{|\mathcal{V}|}{k}$ sequences of items of length k.

3 The EdGe Algorithm

In this paper, we develop EdGe, an Edge-based Greedy sequence recommendation algorithm, which takes as input a directed graph and outputs a sequence of at most k items. EdGe essentially extends the classical greedy algorithm proposed in [6], but instead of selecting at each step the item (node) with a maximum gain, it selects the next valid edge with a maximum *gain according to the edges that are selected so far*. More formally, we select at each step the edge e with the maximal obtained gain, i.e. the edge e maximizing $h(edges(\sigma) \cup e)$, according to the sequence of items σ that is selected so far.

EdGe (pseudo-code in Algorithm 1), takes as input a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a maximum length k of the sequence to be recommended, a submodular monotone function $h(.)$ over the set of edges $\mathcal{E}$. EdGe outputs a sequence of items of length at most k. EdGe starts with the empty sequence σ (line 1). At each step, it computes the set of valid edges C (line 3). An edge e is said to be valid if it preserves the order of the recommended sequence so far, i.e., its endpoint node is not already in the sequence σ, which preserves the order of the corresponding sequence according to the topological ordering of the graph $\mathcal{G}$, and if adding the items induced by this edge to the actual sequence does not violate the maximum cardinality k. If the set C is not empty, EdGe selects the edge e_{ij} with the maximum gain (line 6). Note that the computation of $h()$ is incremental. It is not recomputed from scratch at every iteration. If the selected edge is a self-loop, EdGe appends a single node v_j to the sequence σ, if the start point of the selected edge is already in the sequence σ, it also appends the single node v_j (line 7,8). Finally, if the selected edge e_{ij} induces two different items i and j and item i does not appear in the sequence σ, it appends node v_i

followed by node v_j to the sequence σ (line 10). The algorithm stops when it is not possible to select any further edge to grow the sequence σ (line 4).

Our algorithm does not need full sorting at each step compared to OMEGA [15]. It does a pseudo-sort while computing the set of valid edges and grows the set of selected edges, i.e., it discards all edges whose endpoints nodes are already in the selected sequence so far, to ensure that the recommended sequences will always be sorted according to the topological ordering of the graph.

Algorithm 1: EDGE-BASED GREEDY SEQUENCE ALGORITHM (EDGE)

Input: a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a maximum length of the sequence k, a submodular monotone function $h(.)$ over the set of edges $\mathcal{E}$
Output: Recommended sequence with cardinality at most k

1 $\sigma \leftarrow \emptyset$
2 **while** $|\sigma| \leq k$ **do**
3 $\quad C \leftarrow \{e_{ij} \in \mathcal{E}\ , |nodes(edges(\sigma) \cup e_{ij})| \leq k\ \ \&\ \ v_j \notin \sigma\}$
4 $\quad$ **if** $C = \emptyset$ **then**
5 $\quad\quad$ break
6 $\quad e_{ij} \leftarrow argmax_{e \in C}\ \ h(edges(\sigma) \cup e)$
7 $\quad$ **if** $(i = j) \vee\ (v_i \in \sigma)$ **then**
8 $\quad\quad \sigma \leftarrow \sigma\ \oplus\ v_j$
9 $\quad$ **else**
10 $\quad\quad \sigma \leftarrow \sigma\ \oplus (\ v_i\ \oplus v_j)$
11 **return** σ

3.1 Comparison with OMEGA Algorithm

In previous work on recommending item sequences [15], an algorithm (OMEGA) was proposed. OMEGA iteratively and greedily extends a set of edges E with the corresponding sequence of items σ. The algorithm is based on the assumption that the graph $\mathcal{G}$ is a DAG (directed acyclic graph). At each step, OMEGA selects an edge and reorders the sequence of items induced by the edges selected so far according to the topological ordering of the DAG graph $\mathcal{G}$. OMEGA has a runtime complexity of $\mathcal{O}(m+n+k\Delta m(k \cdot log(k)))$. Where m is the number of edges in the graph, n the number of nodes and $\Delta = min(\Delta_{in}, \Delta_{out})$, where Δ_{in}, Δ_{out} are the indegree and outdegree of the graph. The first term $(n+m)$ is the runtime complexity for computing a topological sort of the graph. In the second term, $k\Delta$ results from the fact that for a sequence of length k, the algorithm can select at most $k\Delta$ edges, and the factor $k \cdot log(k)$ is the complexity for sorting the sequence according to the topological ordering induced by the graph.

The bottleneck of the OMEGA algorithm is that at each step where an edge is selected to greedily extend the set of edges and the corresponding set of items , the algorithm requires a full reordering of the corresponding items that are

induced by all the selected edges so far according to the topological ordering of the input graph.

Our EdGe algorithm builds on the same approach of greedily selecting and extending a set of edges and the corresponding sequence of items. But without the requirement of reordering the sequence at each step. Instead, our approach considers only those candidate edges that are not violating the ordering of the greedily selected sequence. At each iteration, we heuristically discard all edges having their endpoints nodes in the selected sequence so far. This process insures that the final recommended sequence will always be sorted according to the topological ordering of the graph. Furthermore, our EdGe algorithm has the advantage that it does not require the graph $\mathcal{G}$ to be a DAG, as it does not compute an ordering of the graph. This may be beneficial if we consider application scenarios where it is possible to recommend the same item multiple times in the sequence.

3.2 Runtime Complexity

The computational complexity of EdGe is $\mathcal{O}(m \cdot k)$, where m is the number of edges in the graph and k is the maximum length of the recommended sequence. Indeed, selecting the set of valid edges C and retrieving the edge with the maximum gain requires at worse to loop through all the m edges in the graph. This has to be done at most k times since k is the maximum length of the recommended sequence. This complexity is better than OMEGA's [15] whose computational complexity is $\mathcal{O}(m + n + k\Delta m(k \cdot log(k)))$.

4 Experiments

We run experiments on synthetic and real datasets to study the performance of EdGe in terms of response time, achieved value of the objective function and recommendation precision. Our implementation is in Python 3.7.0 and is running on a 2.7 GHz Intel Core i7 machine with a 16 GB main memory, running OS X 10.13.6.

Table 1. Experimental parameters (default values in bold)

Description	Parameter	Values
Number of nodes	n	10, 50, **1000**, 5000, 10000
Maximum outdegree	Δ_{out}	1, 2, 3, 4, 5, **6**, 7, 8, 9, 10
Sequence length	k	1, 5, **10**, 25, 50, 75, 100

4.1 Synthetic Datasets

For this experiment, we follow the same setting as in [15]. We create the input graph $\mathcal{G}$ as follows: let $\mathcal{A}$ be the adjacency matrix of $\mathcal{G}$, i.e. $A_{ij} = 1$ if there

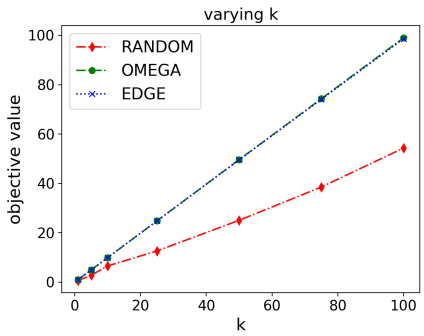

Fig. 2. varying sequence length k

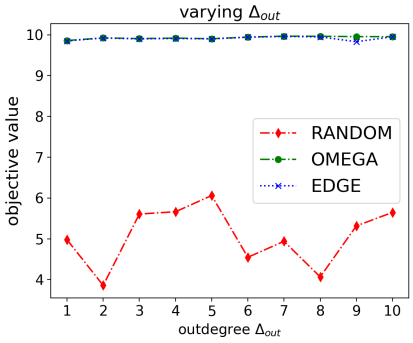

Fig. 3. varying outdegree Δ_{out}

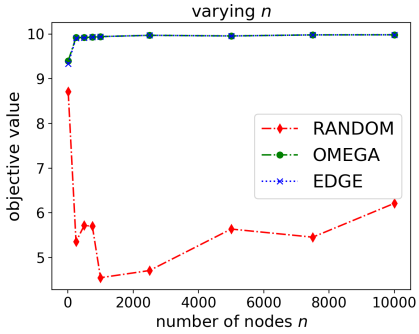

Fig. 4. varying number of nodes n

exists an edge from item i to item j. For every $i \in [n]$ we select a subset of size $min(\Delta_{out}, n-i)$ uniformly at random from $[i+1, n]$ and set the corresponding entries of the matrix A to 1. This construction process of the graph $\mathcal{G}$ ensures a desired maximum outdegree Δ_{out}. It also guarantees that the input graph is a DAG which lets us compare EdGe with OMEGA [15]. We assign every edge of $\mathcal{G}$ a utility value that is independently drawn from a uniform distribution in $\mathcal{U}(0,1)$. We then use those utilities to define the submodular function $h(.)$ as follows:

$$h(E) = \sum_{j \in nodes(E)} [1 - \prod_{(i,j) \in E} (1 - w_{i,j})] \tag{4}$$

where $w_{i,j}$ is the utility associated with edge (i,j) and $nodes(E)$ is the set of items that are induced by the set of edges E.

In the following, we report the values of the objective function for solutions computed by EdGe and by OMEGA. We also use as a baseline a random selection of k items. In each setting, we compare the objective values for different possible combinations of the parameters: k (sequence length), n (number of nodes) and Δ_{out} (maximum outdegree). Table 1 summarizes those parameters.

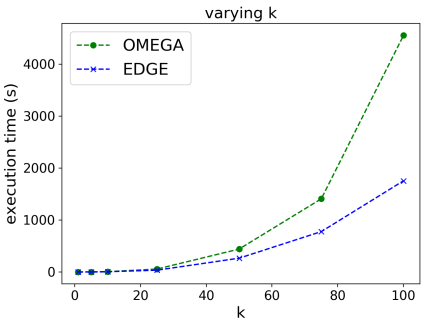

Fig. 5. varying sequence length k

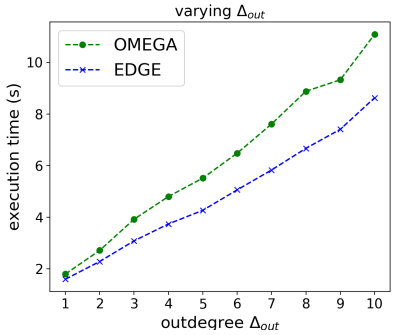

Fig. 6. varying outdegree Δ_{out}

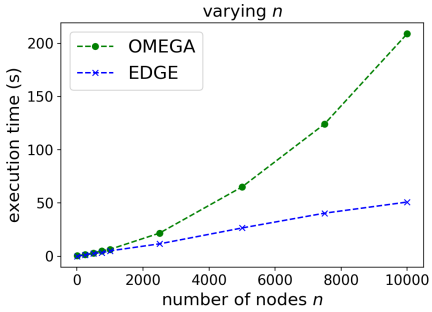

Fig. 7. varying number of nodes n

In each run, we vary one parameter and set the others to their default values which are written in bold.

Results are shown in Figures 2, 3 and 4. For varying the length sequence k, the maximum outdegree Δ_{out} and the number of nodes n, we can observe that EdGe performs very closely to OMEGA according to the achieved value of the objective function f, while being much faster. Results on runtime according to each setting are reported in Figures 5, 6 and 7. We notice that EdGe runs approximatively 3 times faster than OMEGA when $k = 100$, and 4 times faster when $n = 10,000$

4.2 Real Datasets

Datasets and Setup We use two real datasets in our experiments: *Movielens 1M*[3] and a dataset extracted from *Foursquare* [19]. The *Movielens 1M* dataset contains $1,000,209$ ratings of $6,040$ users for $3,706$ movies. Every rating takes a value in $\{1, ..., 5\}$ and has a timestamp associated with it. For the *Foursquare*

[3] http://grouplens.org/datasets/movielens/1m/

dataset, in order to make our setup realistic, we choose POIs from a single city: "New York". We excluded POIs that were visited by fewer than 20 users, and excluded users that visited less than 20 POIs. We end up with a dataset having 840 users and 4,750 POIs. Each POI is associated to a category, including restaurants, bars, parks, etc.

In this experiment, for both datasets, we seek to predict the sequence of items consumed by single users. i.e., sequences of rated movies in the case of *Movielens* and sequence of visited POIs in the case of *Foursquare*. In line with the conducted experiments for OMEGA [15], the input data is viewed as a set of sequences, one per user. We used the provided timestamps to create per-user sequences. The data is randomly split into training $\mathcal{D}_{Train}$ and testing $\mathcal{D}_{Test}$ where $|\mathcal{D}_{Test}| = 500$ for *Movielens* and $|\mathcal{D}_{Test}| = 200$ for *Foursquare*. In both cases, we use randomly selected users for our tests. Our task is to recommend a sequence of items to a test user not present in the training data given a few past consumed items of that user. Formally, for a test user u let $\sigma = (\sigma_1, \sigma_2, ..., \sigma_m)$ be the sequence of items consumed by the user u. Then, given the first half of these items, i.e. $\sigma^{previous} = (\sigma_1, \sigma_2, ..., \sigma_l)$, where $l = \frac{m}{2}$, our goal is to make predictions about which other items the user consumed later, i.e. $\sigma^{left} = (\sigma_{l+1}, \sigma_{l+2}, ..., \sigma_m)$.

Given the length k of the recommended sequence for each user in the training set, $Prec@k$ is the number of correct prediction at k averaged over the test set:

$$Prec@k = \frac{1}{k.|\mathcal{D}_{Test}|}. \sum_{\sigma \in \mathcal{D}_{Test}} |nodes(\sigma^{left}) \cap P_k(\sigma^{previous})| \tag{5}$$

where, $P_k(\sigma^{previous})$ is the predicted sequence of items.

Baseline For both datasets, we compare the results of EdGe to OMEGA [15]. For a fair comparison, we conduct the experiments using the same input graph model that is shown in Figure 8. In this graph, solid nodes represent the items in $\sigma^{previous} = (\sigma_1, \sigma_2, ..., \sigma_l)$, empty nodes represent the candidate items. We model the dependencies between the last z items in $\sigma^{previous}$ and the items that can be selected, where $z = \{1, 2, 5\}$ (for example, $z = 2$ means that the 2 last items in $\sigma^{previous}$ are considered). Note that, the input graph is test instance dependent, i.e., we construct one graph per test user.

The value associated with an edge linking item i to item j noted as $p_{j|i}$ is estimated as the conditional probability that a user consumes j given that she has consumed i before. These conditional probabilities are estimated using the training data. The value associated with every self-loop edge (i, i) corresponds to the empirical frequency p_i of item i (which is also conveniently noted as $p_{i|i}$). These conditional probabilities are then used for estimating the utility function:

$$h(E) = \sum_{j \in nodes(E)} \left[1 - \prod_{(i,j) \in E} (1 - p_{j|i}) \right] \tag{6}$$

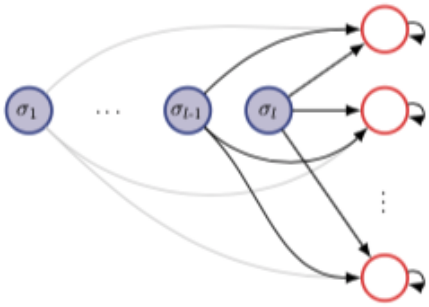

Fig. 8. Input graph [15]. Solid nodes represent the items in $\sigma^{previous} = (\sigma_1, \sigma_2, ..., \sigma_l)$, empty nodes represent the items that can be recommended.

Table 2. MovieLens: OMEGA vs EdGe for Precision ($Prec@k$)

	z=1		z=2		z=5	
	OMEGA	**EdGe**	OMEGA	**EdGe**	OMEGA	**EdGe**
k=1	0.346	**0.346**	0.37	**0.37**	0.38	**0.38**
k=2	0.331	**0.331**	0.359	**0.349**	0.374	**0.345**
k=3	0.3166	**0.3166**	0.346	**0.3406**	0.3699	**0.3330**
k=4	0.3045	**0.3045**	0.3385	**0.327**	0.3665	**0.324**
k=5	0.2916	**0.2916**	0.3304	**0.316**	0.3712	**0.314**

Table 3. Foursquare: OMEGA vs EdGe for Precision ($Prec@k$)

	z=1		z=2		z=5	
	OMEGA	**EdGe**	OMEGA	**EdGe**	OMEGA	**EdGe**
k=1	0.426	**0.426**	0.513	**0.513**	0.70	**0.70**
k=2	0.42	**0.416**	0.518	**0.49**	0.683	**0.678**
k=3	0.404	**0.398**	0.496	**0.474**	0.688	**0.667**
k=4	0.382	**0.38**	0.481	**0.472**	0.665	**0.658**
k=5	0.372	**0.37**	0.478	**0.47**	0.641	**0.637**

Table 4. MovileLens: OMEGA vs EdGe (average runtime in milliseconds)

	z=1		z=2		z=5	
	OMEGA	**EdGe**	OMEGA	**EdGe**	OMEGA	**EdGe**
k=1	72.3	**55.26**	106.16	**79.5**	207.04	**151.9**
k=2	170.6	**116.64**	263	**171.74**	541.32	**334.54**
k=3	306.32	**192.44**	514.04	**284.62**	1087.2	**569.24**
k=4	484.02	**304.62**	836.88	**442.14**	1894.6	**861.14**
k=5	729.3	**410.7**	1240.04	**620.88**	3030.26	**1253.44**

Results Recommendation precision for EdGe and OMEGA is reported in Table 2 for *MovieLens* and Table 3 for *Foursquare*. We vary k, the length of the

Table 5. Foursquare: OMEGA vs EdGe (average runtime in milliseconds)

	z=1		z=2		z=5	
	OMEGA	**EdGe**	OMEGA	**EdGe**	OMEGA	**EdGe**
k=1	90.9	**56.5**	133.15	**79.65**	258.95	**147.35**
k=2	226.05	**115.15**	356.3	**167.95**	718.1	**328.15**
k=3	417.3	**198.15**	699.35	**286.35**	1454.35	**579.1**
k=4	677	**294.8**	1121.75	**439.2**	2474.4	**907.1**
k=5	977.25	**424.9**	1653.9	**661.45**	3778.75	**1285.55**

predicted sequences and z the length of the user's history considered for constructing the input graph. For both experiments, we observe that EdGe performs very closely to OMEGA and equally for some settings. We also notice that the precision increases with increasing z, indicating that considering a longer history of user interactions is beneficial. We also observe that EdGe is much faster than OMEGA, especially when increasing the length of the recommended sequences. Results on the average runtime per user are reported in Table 4 for *MovieLens* and in Table 5 for *Foursquare*.

5 Related Work

In [12], the authors propose an interactive itinerary planning for ranking travel packages. However, in this work the order of visiting POIs is mainly due to the POI visit and transit times induced by the recommended itinerary. The utility of selecting POIs in a certain order is neglected and the system does not take into account ordered preferences. Some existing works address the problem of package recommendations [1, 17]. However, in such works, authors assign a utility for sets of items but not for sequences, i.e., sequential dependencies are neglected. Another research area which deals with the problem of selecting sequences is sequential pattern mining. This problem was introduced by Agrawal and Srikant [14]. Sequential pattern mining is the task of finding all frequent subsequences in a sequence database. Numerous algorithms have been designed to discover sequential patterns in sequence databases, such as FPGrowth [2], Spade [21] and Prefixspan [9]. However, these works do not take into account user preferences and output the same subsequences for every user which leads to a lack of personalization.

Another related research area is *next basket recommendation* [10, 20] which has gained much attention especially in e-commerce scenarios. Two different approaches were used to tackle this problem, the well-known collaborative filtering (CF) models such as Matrix factorization (MF) [4] which capture user's general interests but have a lack to consider sequential behaviors, and Markov chain models which extract sequential features from the history on users' interactions and then predict next purchase, but have lack to consider general preferences of the user. Rendle *et al* [10] combined the well-known collaborative filtering (CF) using Matrix factorization (MF) [4] and Markov chain models which ex-

tract sequential features from the history on users' interactions to account for both users' interests and sequential features in order to generate ranking lists. However, these works are different from the one we proposed in this paper. They don't formalize the optimization problem of recommending a sequence of items by modeling the sequential dependencies with a graph data model.

Another related work is the recommendation with prerequisites on items, which was studied in [7]. Authors addressed the problem of recommending the best set of k items when there is an inherent ordering between items expressed as prerequisites. Various prerequisite structures were studied and several heuristic approximation algorithms were developed to solve the problem. However, the problem is formulated as a set recommendation problem rather than a sequential recommendation problem.

Our formalization of sequence recommendation builds on the one proposed recently in [15]. We show that our algorithm, EdGe produces recommendations that are as good and has much better runtime complexity and response time.

6 Conclusion

We set out to solve the sequence recommendation problem. We proposed a submodular maximization formulation that is based on marginal edge utility, and developed an efficient greedy algorithm to find the best sequence to recommend. We validated our results on synthetic and real datasets and showed that EdGe, our Edge-based Greedy sequence recommendation algorithm, returns recommendation with high accuracy in little time. This work opens several directions. Our immediate improvement is to study the addition of constraints to the objective function including constraints on the type of items to consider and on the types of transitions between items. Additionally, we would like to examine how to simultaneously recommend sequences to multiple users. We also plan to test our solutions on other real datasets acquired from industry partners.

References

1. Amer-Yahia, S., Bonchi, F., Castillo, C., Feuerstein, E., Mendez-Diaz, I., Zabala, P.: Composite retrieval of diverse and complementary bundles. IEEE Transactions on Knowledge and Data Engineering **26**(11), 2662–2675 (2014)
2. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data mining and knowledge discovery **8**(1), 53–87 (2004)
3. Hariri, N., Mobasher, B., Burke, R.: Context-aware music recommendation based on latenttopic sequential patterns. In: Proceedings of the Sixth ACM Conference on Recommender Systems. pp. 131–138. RecSys '12, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2365952.2365979, http://doi.acm.org/10.1145/2365952.2365979
4. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer (8), 30–37 (2009)

5. Krause, A., Golovin, D.: Submodular Function Maximization, p. 71–104. Cambridge University Press (2014). https://doi.org/10.1017/CBO9781139177801.004
6. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions—i. Mathematical programming **14**(1), 265–294 (1978)
7. Parameswaran, A., Venetis, P., Garcia-Molina, H.: Recommendation systems with complex constraints: A course recommendation perspective. ACM Transactions on Information Systems (TOIS) **29**(4), 20 (2011)
8. Pazzani, M.J., Billsus, D.: Content-based recommendation systems. In: The adaptive web, pp. 325–341. Springer (2007)
9. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Transactions on knowledge and data engineering **16**(11), 1424–1440 (2004)
10. Rendle, S., Freudenthaler, C., Schmidt-Thieme, L.: Factorizing personalized markov chains for next-basket recommendation. In: Proceedings of the 19th international conference on World wide web. pp. 811–820. ACM (2010)
11. Ricci, F., Rokach, L., Shapira, B.: Recommender systems: introduction and challenges. In: Recommender systems handbook, pp. 1–34. Springer (2015)
12. Roy, S.B., Das, G., Amer-Yahia, S., Yu, C.: Interactive itinerary planning. In: Data Engineering (ICDE), 2011 IEEE 27th International Conference on. pp. 15–26. IEEE (2011)
13. Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S.: Collaborative filtering recommender systems. In: The adaptive web, pp. 291–324. Springer (2007)
14. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: International Conference on Extending Database Technology. pp. 1–17. Springer (1996)
15. Tschiatschek, S., Singla, A., Krause, A.: Selecting sequences of items via submodular maximization. In: AAAI. pp. 2667–2673 (2017)
16. Wörndl, W., Hefele, A., Herzog, D.: Recommending a sequence of interesting places for tourist trips. Information Technology & Tourism **17**(1), 31–54 (2017)
17. Xie, M., Lakshmanan, L.V., Wood, P.T.: Breaking out of the box of recommendations: from items to packages. In: Proceedings of the fourth ACM conference on Recommender systems. pp. 151–158. ACM (2010)
18. Xu, J., Xing, T., Van Der Schaar, M.: Personalized course sequence recommendations. IEEE Transactions on Signal Processing **64**(20), 5340–5352 (2016)
19. Yang, D., Zhang, D., Zheng, V.W., Yu, Z.: Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. IEEE Transactions on Systems, Man, and Cybernetics: Systems **45**(1), 129–142 (2015)
20. Yu, F., Liu, Q., Wu, S., Wang, L., Tan, T.: A dynamic recurrent model for next basket recommendation. In: Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval. pp. 729–732. ACM (2016)
21. Zaki, M.J.: Spade: An efficient algorithm for mining frequent sequences. Machine learning **42**(1-2), 31–60 (2001)