Exact-K Recommendation via Maximal Clique Optimization

Yu Gong[1,*], Yu Zhu[1,*], Lu Duan[2], Qingwen Liu[1], Ziyu Guan[3], Fei Sun[1], Wenwu Ou[1], Kenny Q. Zhu[4]
[1] Alibaba Group, China [2] Zhejiang Cainiao Supply Chain Management Co., Ltd, China
[3] Xidian University, China [4] Shanghai Jiao Tong University, China
{gongyu.gy,zy143829,xiangsheng.lqw,ofey.sf}@alibaba-inc.com,duanlu.dl@cainiao.com
zyguan@xidian.edu.cn,santong.oww@taobao.com,kzhu@cs.sjtu.edu.cn

ABSTRACT

This paper targets to a novel but practical recommendation problem named exact-K recommendation. It is different from traditional top-K recommendation, as it focuses more on (constrained) combinatorial optimization which will optimize to recommend a whole set of K items called card, rather than ranking optimization which assumes that "better" items should be put into top positions. Thus we take the first step to give a formal problem definition, and innovatively reduce it to Maximum Clique Optimization based on graph. To tackle this specific combinatorial optimization problem which is NP-hard, we propose *Graph Attention Networks* (GAttN) with a Multi-head Self-attention encoder and a decoder with attention mechanism. It can end-to-end learn the joint distribution of the K items and generate an optimal card rather than rank individual items by prediction scores. Then we propose *Reinforcement Learning from Demonstrations* (RLfD) which combines the advantages in behavior cloning and reinforcement learning, making it sufficient-and-efficient to train the model. Extensive experiments on three datasets demonstrate the effectiveness of our proposed *GAttN with RLfD* method, it outperforms several strong baselines with a relative improvement of 7.7% and 4.7% on average in Precision and Hit Ratio respectively, and achieves state-of-the-art (SOTA) performance for the exact-K recommendation problem.

CCS CONCEPTS

• **Information systems** → **Learning to rank**; **Recommender systems**.

KEYWORDS

recommender system; exact-K recommendation; learning-to-rank; reinforcement learning; encoder-decoder

ACM Reference Format:
Yu Gong, Yu Zhu, Lu Duan, Qingwen Liu, Ziyu Guan, Fei Sun, Wenwu Ou, Kenny Q. Zhu. 2019. Exact-K Recommendation via Maximal Clique Optimization. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'19), June 22–24, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3292500.3330832

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330832

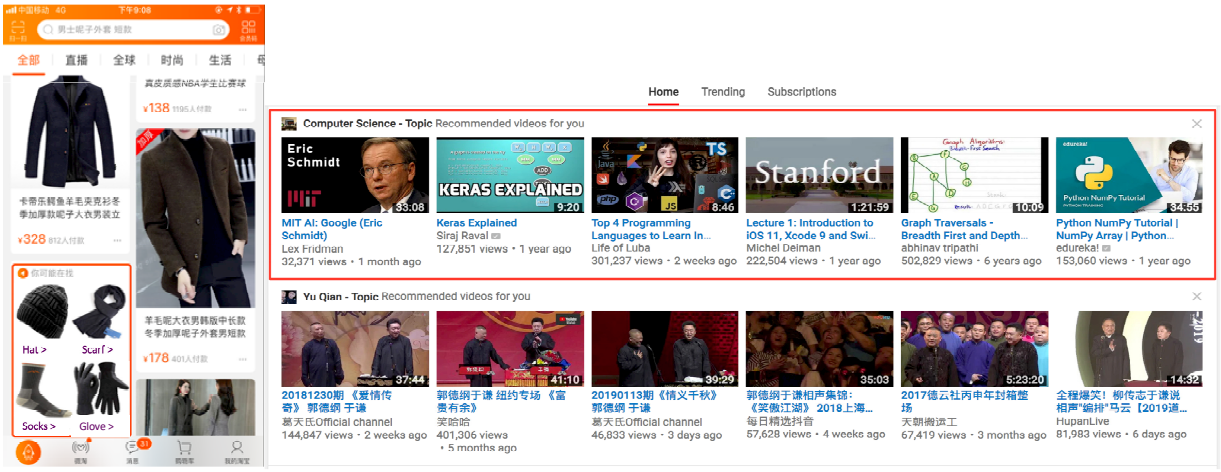
1 INTRODUCTION

The explosive growth and variety of information (e.g. movies, commodities, news etc.) available on the web frequently overwhelms users, while Recommender Systems (RS) are valuable means to cope with the information overload problem. RS usually provide the target user with a list of items, which are selected from the overwhelmed candidates in order to best satisfy his/her current demand. In the most traditional scenarios of RS especially on mobiles, recommended items are shown in a waterfall flow form, i.e. users should scroll the screen and items will be presented one-by-one. Due to the pressure of QPS (Query-Per-Second) for users interacting with RS servers, it is common to return a large amount of ranked items (e.g. 50 items in Taobao RS) based on CTR (Click-Through-Rate) estimation for example[1] and present them from top to bottom. That is to say we believe the top ranked items take the most chance to be clicked or preferred so that when users scroll the screen and see items top-down, the overall clicking efficiency can be optimized. It can be seen as *top-K* recommendation [7], because the ranking of item list is important.

However in many real-world recommendation applications, exact K items are shown once all to the users. In other words, users should not scroll the screen and the combination of K items is shown as a whole **card**. Taking two popular RS in the homepages of Taobao and YouTube for example (illustrated in Fig. 1), they recommend cards with exact 4 commodities and 6 videos respectively. Note that items in the same card may interact with each other, e.g. in Taobao, co-occurrence of "hat" and "scarf" performs better than "shoe" and "scarf", but "shoe" and "scarf" can be optimal individually. We call it *exact-K* recommendation, whose key challenge is to maximize the chance of the whole card being clicked or satisfied by the target user. Meanwhile, items in a card usually maintain some **constraints** between each other to guarantee the user experience in RS, e.g. the recommended commodities in E-commerce should have some diversity rather than being all similar for complement consideration. In a word, top-K recommendation can be seen as a ranking optimization problem which assumes that "better" items should be put into top positions, while exact-K recommendation is a (constrained) combinatorial optimization problem which tries to maximize the joint probability of the set of items in a card.

Top-K recommendation has been well studied for decades in information retrieval (IR) research community. Among them, listwise models are the most related to our problem as they also perform optimization considering the whole item list. However, they either target on ranking refinement or do not consider constraints in the

[*] Equal contribution.
[1] Here we take CTR as an example, other preference score can also be used, e.g. Movie Rating, CVR (Conversion-Rate) or GMV (Gross-Merchandise-Volume) etc.



(a) Taobao RS (b) YouTube RS

Figure 1: Show cases for exact-K recommendation in Taobao and YouTube.

ranking list, which will fall into sub-optimal towards exact-K recommendation (refer to Sec. 2.1.2 for more discussions). Our work mainly focuses on solving exact-K recommendation problem end-to-end, and its main contributions can be summarized as follows.

(1) We take the first step to formally define the exact-K recommendation problem and innovatively reduce it to a Maximal Clique Optimization problem based on graph.
(2) To solve it, we propose *Graph Attention Networks* (GAttN) with an Encoder-Decoder framework which can end-to-end learn the joint distribution of K items and generate an optimal card containing K items. Encoder utilizes Multi-head Self-attention to encode the constructed undirected graph into node embeddings considering nodes correlations. Based on the node embeddings, decoder generates a clique consisting of K items with RNN and attention mechanism which can well capture the combinational characteristic of the K items. Beam search with masking is applied to meet the constraints. Then we adopt well-designed *Reinforcement Learning from Demonstrations* (RLfD) which combines the advantages in behavior cloning and reinforcement learning, making it sufficient-and-efficient to train GAttN.
(3) We conduct extensive experiments on three datasets (two constructed from public MovieLens datasets and one collected from Taobao). Both quantitative and qualitative analysis justify the effectiveness and rationality of our proposed *GAttN with RLfD* for exact-K recommendation. Specifically, our method outperforms several strong baselines with significant improvements of 7.7% and 4.7% on average in Precision and Hit Ratio respectively.

2 RELATED WORKS

2.1 Top-K Recommendation

Top-K recommendation refers to recommending a list of K ranked items to a user, which is related to the descriptions of recommendation problem and learning to rank methods.

2.1.1 The Recommendation Problem. The key problem of recommendation system lies in how to generate users' most preferred item list. Some previous works [17] model the recommendation problem as a regression task (i.e. predict users' ratings on items) or classification task (i.e. predict whether the user will click/purchase/...the item). Items are then ranked based on the regression scores or classification probabilities to form the recommendation list. Other works [14, 22, 29] directly model the recommendation problem as a ranking task, where many pairwise/listwise ranking methods are exploited to generate users' top-k preferred items. Learning to rank is surveyed in detail in the next subsection.

2.1.2 Learning to Rank. Learning to Rank (LTR) refers to a group of techniques that attempts to solve ranking problems by using machine learning algorithms. It can be broadly classified into three categories: pointwise, pairwise, and listwise models. Pointwise Models [14, 17] treat the ranking task as a classification or regression task. However, pointwise models do not consider the inter-dependency among instances in the final ranked list. Pairwise Models [22] assume that the relative order between two instances is known and transform it to a pairwise classification task. Note that their loss functions only consider the relative order between two instances, while the position of instances in the final ranked list can hardly be derived. Listwise Models provide the opportunity to directly optimize ranking criteria and achieve whole-page ranking optimization. Recently [2] proposed Deep Listwise Context Model (DLCM) to fine-tune the initial ranked list generated by a base model, which achieves SOTA performance. Other whole-page ranking optimization methods can be found in [16], which mainly focus on ranking refinement. Listwise models are the most related to our problem. However, they either target on ranking refinement or don't consider the constraints in ranking list, which are not well-designed for exact-K recommendation.

2.2 Neural Combinatorial Optimization

Even though machine learning (ML) and combinatorial optimization have been studied for decades respectively, there are few investigations on the application of ML methods in solving the combinational optimization problem. Current related works mainly focus on two types of ML methods: supervised learning and reinforcement

learning. Supervised learning [27] is the first successful attempt to solve the combinatorial optimization problem. It proposes a special attention mechanism named Pointer-Net to tackle a classical combinational optimization problem: Traveling Salesman Problem (TSP). Reinforcement learning (RL) aims to transform the combinatorial optimization problem into a sequential decision problem and becomes increasingly popular recently. Based on Pointer Network, [5] develops a neural combinatorial optimization framework with RL, which performs excellently in some classical problems, such as TSP and Knapsack Problem. RL is also applied to RS [32], but it is still designed for traditional top-K recommendation. In this work, we focus on exact-K recommendation, which is transferred into the maximal clique optimization problem. Some researches [15] also try to solve it, but they often focus on estimation of node-weight. The main difference between them and ours is that we target to directly select an optimal clique rather than search for the clique comprised of maximum weighted nodes, which brings a grave challenge.

3 PROBLEM DEFINITION

In this section, we first give a formal definition of exact-K recommendation, and then discuss how to transfer it to the Maximal Clique Optimization problem. Finally we provide a baseline approach to tackle the above problem.

3.1 Exact-K Recommendation

Given a set of candidate N items $S = \{s_i\}_{1 \le i \le N}$, our goal is to recommend exact K items $A = \{a_i\}_{1 \le i \le K} \subseteq S$ which is shown as a whole card[2], so that the combination of items A takes the most chance to be clicked or satisfied by a user u. We denote the probability of A being clicked/satisfied as $P(A, r = 1|S, u)$. Somehow items in A should obey some M constraints between each other as $C = \{c_k(a_i, a_j) = 1 | a_i \in A, a_j \in A, i \neq j\}_{1 \le k \le M}$ or not as $C = \emptyset$, here c_k is a boolean indicator which will be 1 if the two items satisfy the constraint. Overall the problem of exact-K recommendation can be regarded as a (constrained) combinatorial optimization problem, and is defined formally as follows:

$$\max_{A} P(A, r = 1|S, u; \theta), \tag{1}$$

$$s.t\ \forall a_i \in A, a_j \in A, i \neq j, \forall c_k \in C, c_k(a_i, a_j) = 1, \tag{2}$$

where θ is the parameters for function of generating A from S given user u, and $r = 1$ donates relevance/preference indicator.

In another perspective, we construct a graph $\mathbb{G}(\mathcal{N}, \mathcal{E})$ containing N nodes, in which each node n_i in $\mathcal{N}$ represents an item s_i in candidate item set S, each edge e_{ij} in $\mathcal{E}$ connecting nodes (n_i, n_j) represents that items s_i and s_j should satisfy the constraints or there is no constraint (a.k.a $\mathbb{G}$ is now a complete graph), i.e $\forall c_k \in C, c_k(s_i, s_j) = 1$ or $C = \emptyset$, so it is an undirected graph here. Intuitively, we can transfer exact-K recommendation into the maximal clique optimization problem[3] [9, 12]. That is to say we aim to select a clique[4] (i.e characteristics of clique can ensure the constraints defined in Eq. 2) with K nodes from $\mathbb{G}$ where combination of the selected K corresponding items A achieves the maximal objective defined in Eq. 1. You can take Fig. 2 as an example. Furthermore, maximal clique problem is proved to be NP-hard thus it can not be solved in polynomial time [9].

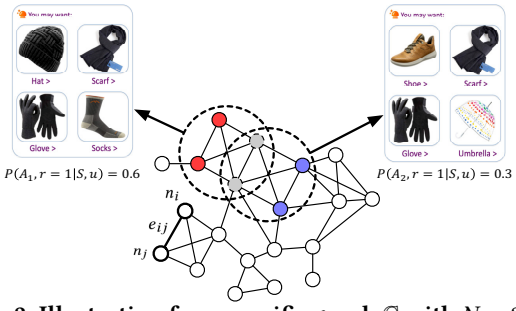


Figure 2: Illustration for a specific graph $\mathbb{G}$ with $N = 20$ and $K = 4$. We show two different cliques (red and blue) in graph and the corresponding cards (A_1 and A_2) each with 4 items. We suppose that $P(A_1, r = 1|S, u) > P(A_2, r = 1|S, u)$ means that card A_1 takes more chance to be satisfied than card A_2 given user u and candidate item set S.

3.2 Naive Node-Weight Estimation Method

A baseline method is that we can estimate a weight as w_i of each node $n_i \in \mathcal{N}$ related to the optimization objective in graph $\mathbb{G}$. In exact-K recommendation, our goal is to maximize the clicked or satisfied probability of the recommended K items set as in Eq. 1, so we regard the weight of each node as CTR of corresponding item. After getting the weight of each node in graph supported as $\mathbb{G}(\mathcal{N}, \mathcal{E}, \mathcal{W})$, we can reduce the Maximal Clique Optimization problem as finding a clique in graph $\mathbb{G}$ with maximal node weights summation. We can then apply some heuristic methods like Greedy search to solve it. Specifically, we modify Eq. 1 as follows:

$$\max_{A} \sum_{a_i \in A} P(r = 1|a_i, S, u; \theta), \tag{3}$$

where $P(r = 1|a_i, S, u; \theta)$ can be regarded as node weight w_i in graph. Here we focus on how to estimate the node weights $\mathcal{W}$, it can be formulated as a normal item CTR estimation task in IR. A large amount of LTR based methods for CTR estimation can be adopted as our strong baselines. Refer to Sec. 2.1.2 for more details.

We call the adapted baseline as Naive Node-Weight Estimation Method, with its detailed implementation shown in Algorithm 1 [5]. However weaknesses of such method are obvious for the following three points: (1) CTR estimation for each item is independent, (2) combinational characteristic of the K items in a card is not considered, (3) problem objective is not optimized directly but substituted with a reduced heuristic objective which will unfortunately fall into sub-optimal. On the contrary, we will utilize a framework of Neural Combinatorial Optimization (some related works in Sec. 2.2) to directly optimize the problem objective in Sec. 4.

4 APPROACH

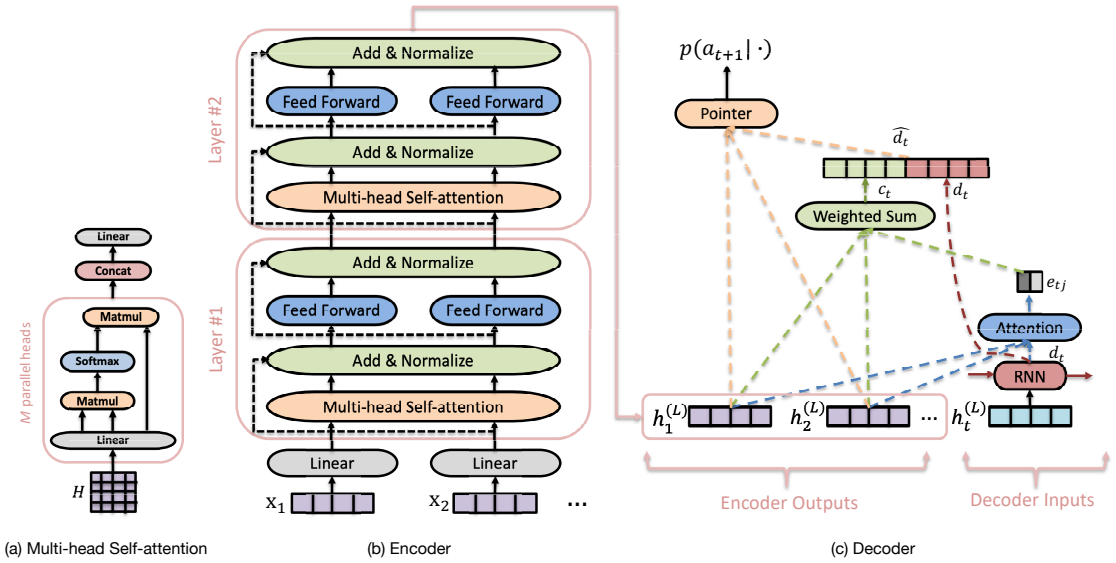
In Sec. 3.1, we formally define the exact-K recommendation problem based on searching for maximal scoring clique with K nodes in a specially constructed graph $\mathbb{G}(\mathcal{N}, \mathcal{E})$ with N nodes. The score

[2]Here we suppose that permutation of the K items in a card is not considered in exact-K recommendation.

[3]It can be generalized according to the optimization objective.

[4]A clique is a subset of nodes of an undirected graph such that every two distinct nodes in the clique are adjacent; that is, its induced subgraph is complete.

[5]In our problem, we ignore the circumstance of getting infeasible solution, and we argue that in real-world application with small K and large N we can always find a clique with K nodes in graph with N nodes greedily.



(a) Multi-head Self-attention (b) Encoder (c) Decoder

Figure 3: The key modules of Graph Attention Networks (GAttN).

of a clique is the overall probability of a user clicking or stratifying the corresponding card of K items as in Eq. 1. To tackle this specific problem, we first propose *Graph Attention Networks* (GAttN) which follows the Encoder-Decoder Pointer framework [27] with Attention mechanism [4, 25]. Then we adopt *Reinforcement Learning from Demonstrations* (RLfD) which combines the advantages in behavior cloning [24] and reinforcement learning, making it sufficient-and-efficient to train the networks.

4.1 Graph Attention Networks

The traditional encoder-decoder framework [23] usually encodes an input sequence into a fixed vector by a recurrent neural networks (RNN) and decodes a new output sequence from that vector using another RNN. For our exact-K recommendation problem, the encoder produces representations of all input nodes in graph $\mathbb{G}$, and the decoder selects a clique A among input nodes by pointer, in which the constraint of clique is considered by masking.

4.1.1 Input. We first define the input representation of each node in graph $\mathbb{G}(\mathcal{N}, \mathcal{E})$. Specifically in our problem, given candidate items set S and user u, we can represent the input x_i of a node $n_i \in \mathcal{N}$ by combination of the features of corresponding item $s_i \in S$ and user u. Here we use a simple fully connected neural network with nonlinear activation ReLU as:

$$x_i = ReLU(W_I[x_{s_i}; x_u] + b_I), \tag{4}$$

where x_{s_i} and x_u are feature vectors for item s_i and user u (e.g trainable embeddings of corresponding item and user IDs), $[a; b]$ represents the concatenation of vector a and b, W_I and b_I are training parameters for input representation.

4.1.2 Encoder. First of all, since the order of nodes in a undirected graph is meaningless, the encoder network architecture should be permutation invariant, i.e. any permutation of the inputs results in the same output representations. While the traditional encoder usually uses a RNN to convey sequential information, e.g., in text translation the relative position of words must be captured, but it is not appropriate to our case. Secondly, the representation for a node should consider the other nodes in graph, as there can exist some underlying structures in graph that nodes may influence between each other. So it's helpful to represent a node with some attentions to other nodes. As a result, we use a model like Self-attention, it is a special case of attention mechanism that only requires a single sequence to compute its representation. Self-attention has been successfully applied to many NLP tasks up to now [30], here we utilize it to encode the graph and produce nodes representations.

Actually in this paper, the encoder that we use is similar to the encoder used in Transformer architecture by [25] with multi-head self-attention. Fig. 3(b) depicts the computation graph of encoder. From the d_x-dimensional input feature vector x_i for node n_i, the encoder firstly computes initial d_h-dimensional graph node embedding (a.k.a representation) $h_i^{(0)}$ through a learned linear projection with parameters W_E and b_E as:

$$h_i^{(0)} = W_E x_i + b_E. \tag{5}$$

The node embeddings are then updated through L self-attention layers, each consisting of two sub-layers: a multi-head self-attention (MHSA) layer followed by a feed-forward (FF) layer. We denote with $H^{(l)} = \{h_i^{(l)}\}_{1 \le i \le N}$ the graph node embeddings produced by layer l, and the final output graph node embeddings as $H^{(L)}$.

The basic component of MHSA is the scaled dot-product attention, which is a variant of dot-product (multiplicative) attention [18]. Given a matrix of N input d-dimensional embedding vectors $E \in \mathbb{R}^{N \times d}$, the scaled dot-product attention computes the self-attention scores based on the following equation:

$$\text{SelfAttention}(E) = softmax(\frac{EE^T}{\sqrt{d}})E, \tag{6}$$

where $softmax$ is row-wise. More specifically, MHSA sub-layers will employ M attention heads to capture multiple attention information and the results from each head are concatenated followed by a parameterized linear transformation to produce the sub-layer

outputs. Fig. 3(a) shows the computation graph of MHSA. Specifically in layer l, it will operate on the output embedding matrices $H^{(l-1)} \in \mathbb{R}^{N \times d_h}$ from previous layer $l-1$ and produce the MHSA sub-layer outputs $\widehat{H^{(l)}} \in \mathbb{R}^{N \times d_h}$ as:

$$\begin{aligned} \widehat{H^{(l)}} &= [head_1; \ldots; head_M] W_O, \\ \text{where } head_i &= \text{SelfAttention}(H^{(l-1)} W_{hi}), \end{aligned} \tag{7}$$

where M is the number of heads, $W_{hi} \in \mathbb{R}^{d_h \times d_k}$ is parameter for each head, $W_O \in \mathbb{R}^{(Md_k) \times d_h}$ is parameter for linear transformation output, and d_k is the output dimension in each head. In addition to MHSA sub-layers, FF sub-layers consist of two linear transformations with a ReLU activation in between.

$$h_i^{(l)} = W_{F2} ReLU(W_{F1} \widehat{h_i^{(l)}} + b_{F1}) + b_{F2}, \tag{8}$$

where W_{F1}, W_{F2} are parameter matrices, $\widehat{h_i^{(l)}}$ and $h_i^{(l)}$ represent embedding outputs of node n_i in MHSA and FF sub-layers correspondingly. We emphasize that those trainable parameters mentioned above are unique per layer. Furthermore, each sub-layer adds a skip-connection [13] and layer normalization [3].

As a result, *encoder* transforms $x_1, x_2, \cdots, x_N$ (original representations of nodes in graph) in Fig. 3(b) to $h_1^{(L)}, h_2^{(L)}, \cdots, h_N^{(L)}$ (embedding representations of these nodes, considering graph constructure information), which will be used in decoder in Fig. 3(c).

4.1.3 Decoder. For exact-K recommendation problem, the output A represents a clique (card) with K nodes (items) in graph $\mathbb{G}$ that can be interrelated with each other. Recently, RNN [10, 11, 27] has been widely used to map the encoder embeddings to a correlated output sequence, so does in our proposed framework. We call this RNN architecture decoder to decode the output nodes $A = \{a_1, \ldots, a_K\}$. Remember our goal is to optimize $P(A|S, u; \theta)$ defined in Sec. 3.1 (here we omit relevance score of $r = 1$), it is a joint probability and can be decomposed by the chain rule as follows:

$$\begin{aligned} P(A|f(S, u; \theta) &= \prod_{i=1}^{K} p(a_i | a_1, \ldots, a_{i-1}, S, u; \theta) \\ &= \prod_{i=1}^{K} p(a_i | a_1, \ldots, a_{i-1}, f(S, u; \theta_e); \theta_d), \end{aligned} \tag{9}$$

where we represent encoder as $f(S, u; \theta_e)$ with trainable parameters θ_e, and decoder trainable parameters as θ_d. The last term in above Eq. 9 is estimated with RNN by introducing a state vector, d_i, which is a function of the previous state d_{i-1}, and the previous output node a_{i-1}, i.e.

$$p(a_i | a_1, \cdots, a_{i-1}, f(S, u; \theta_e); \theta_d) = p(a_i | d_i, f(S, u; \theta_e); \theta_d), \tag{10}$$

where d_i is computed as follows:

$$d_i = \begin{cases} g(0, 0) & \text{if } i = 1, \\ g(d_{i-1}, a_{i-1}) & \text{otherwise,} \end{cases} \tag{11}$$

where $g(d, a)$ is usually a non-linear function (e.g. cell in LSTM [33] or GRU [34]) that combines the previous state and previous output (embedding of the corresponding node a from encoder) in order to produce the current state.

Decoding happens sequentially, and at time-step $t \in \{1, \ldots, K\}$, the decoder outputs the node a_t based on the output embeddings from encoder and already generated outputs $\{a_{t'}\}_{1 \le t' \le t}$ which are embedded by RNN hidden state d_t. See Fig. 3(c) for an illustration of the decoding process. During decoding, $p(a_t | d_t, f(S, u; \theta_e); \theta_d)$ is implemented by an specific attention mechanism named **pointer** [27], in which it will attend to each node in the encoded graph and calculate the attention scores before applying *softmax* function to get the probability distribution. It allows the decoder to look at the whole input graph $\mathbb{G}(\mathcal{N}, \mathcal{E})$ at any time and select a member of the input nodes $\mathcal{N}$ as the final outputs A.

For notation purposes, let us define decoder output hidden states as $(d_1, \ldots, d_K)$, the encoder output graph node embeddings as $(h_1^{(L)}, \ldots, h_N^{(L)})$. At time-step t, decoder first glimpses [26] the whole encoder outputs, and then computes the representation of decoding up to now together with attention to the encoded graph, denoted as $\widehat{d_t}$ and the equation is as follows:

$$\begin{aligned} e_{tj} = softmax(v_{D1}^T \tanh(W_{D1} d_t + W_{D2} h_j^{(L)})), \; j \in \{1, \ldots, N\}, \\ c_t = \textstyle\sum_{j=1}^{N} e_{tj} h_j^{(L)}, \; \widehat{d_t} = [d_t; c_t], \end{aligned} \tag{12}$$

where W_{D1}, W_{D2} and v_{D1} are trainable parameters. After getting the representation of decoder at time-step t, we apply a specific attentive pointer with masking scheme to generate feasible clique from graph. In our case, we use the following masking procedures: (1) nodes already decoded are not allowed to be pointed, (2) nodes will be masked if disobey the clique constraint rule among the decoded subgraph. And we compute the attention values as follows:

$$u_{tj} = \begin{cases} v_{D2}^T \tanh(W_{D3} \widehat{d_t} + W_{D4} h_j^{(L)}), & \text{otherwise,} \\ -\infty, & \text{if node } n_j \text{ should be masked,} \end{cases} \tag{13}$$

where v_{D2}, W_{D3}, and W_{D4} are trainable parameters. Then *softmax* function is applied to get the pointing distribution towards input nodes, as follows:

$$p(a_t | d_t, f(S, u; \theta_e); \theta_d) = softmax(u_{tj}), \; j \in \{1, \ldots, N\}. \tag{14}$$

We mention that the attention mechanism adopted in Eq. 12 and 13 is following Bahdanau et al [4]. At the period of decoder inference, we apply technique of beam search [28]. It is proposed to expand the search space and try more combinations of nodes in a clique (a.k.a items for a card) to get a most optimal solution.

To summarize, *decoder* receives embedding representations of nodes in graph $\mathbb{G}$ from encoder, and selects clique A of K nodes with attention mechanism. With the help of RNN and beam search, decoder in our proposed GAttN framework is able to capture the combinational characteristics of the K items in a card.

4.2 Reinforcement Learning from Demonstrations

4.2.1 Overall. In our proposed GAttN framework, we represent encoder as $f(S, u; \theta_e)$ which can be seen as **state** $\mathcal{S}$ in RL, and we represent decoder as $P(A|f(S, u; \theta_e); \theta_d) = P(A|\mathcal{S}; \theta_d)$ which can be seen as **policy** $\mathcal{P}$ in RL. A *Reinforcement Learning from Demonstration* (RLfd) agent, possessing both an objective reward signal and demonstrations, is able to combine the best from both fields. This framework is first proposed in domain of Robotics [20]. Learning from demonstrations is much sample efficient and can speed up learning process, leveraging demonstrations to direct what would otherwise be uniform random exploration and thus speed up learning. While the demonstration trajectories may be noisy or sub-optimal, so policy supervised from such demonstrations will be worse too. And learning from demonstrations is not directly targeting the objective which makes the policy fall into

local-minimal. On the other hand, training policy by reinforcement learning can directly optimize the objective reward signal, witch allows such an agent to eventually outperform a potentially sub-optimal demonstrator.

4.2.2 Learning from Demonstrations. Learning from demonstrations can be seen as behavior cloning imitation learning [24], it applies supervised learning for policy (mapping states to actions) using the demonstration trajectories as ground-truth. We collect the ground truth clicked/satisfied cards $A^* = \{a_i^*\}_{1 \le i \le K}$ given user u and candidate items set S as demonstration trajectories and formulated as $P^S_{data}(A^*|S,u)$, we can define the following loss function based on cross entropy $CrossEntropy$ of the generated cards $P(A|S,u;\theta)$ and demonstrated cards $P^S_{data}(A^*|S,u)$.

$$\begin{aligned}\mathcal{L}_S(\theta) &= \sum_{S,u} CrossEntropy(P(A|S,u;\theta), P^S_{data}(A^*|S,u)) \\ &= -\sum_{S,u,A^* \in P^S_{data}} \log P(A^*|S,u;\theta) \\ &= -\sum_{S,u,A^* \in P^S_{data}} \sum_{i=1}^{K} \log p(a_i^*|a_1^*,\cdots,a_{i-1}^*,\mathcal{S};\theta_d) \\ &= -\sum_{S,u,A^* \in P^S_{data}} \sum_{i=1}^{K} \log p(a_i^*|d_i^*,\mathcal{S};\theta_d),\end{aligned} \tag{15}$$

where d_i^* in the last term is state vector estimated by a RNN defined in Eq. 11 with inputs of d_{i-1}^* and a_{i-1}^*. This means that the decoder model focuses on learning to output the next item of the card given the current state of the model AND previous ground-truth items.

SUPERVISE with Policy-sampling. During inference the model can generate a full card A given state $\mathcal{S}$ by generating one item at a time until we get K items. For this process, at time-step t, the model needs the output item a_{t-1} from the last time-step as input in order to produce a_t. Since we don't have access to the true previous item, we can instead either select the most likely one given our model or sample according to it. In all these cases, if a wrong decision is taken at time-step $t-1$, the model can be in a part of the state space that is very different from those visited from the training distribution and for which it doesn't know what to do. Worse, it can easily lead to cumulative bad decisions. We call this problem as discrepancy between training and inference [6].

In our work, we propose *SUPERVISE with Policy-sampling* to bridge the gap between training and inference of policy. We change the training process from fully guided using the true previous item, towards using the generated item from trained policy instead. The loss function for *Learning from Demonstrations* is now as follows:

$$\begin{aligned}\mathcal{L}_S(\theta) &= -\sum_{S,u,A^* \in P^S_{data}} \sum_{i=1}^{K} \log p(a_i^*|a_1,\cdots,a_{i-1},\mathcal{S};\theta_d) \\ &= -\sum_{S,u,A^* \in P^S_{data}} \sum_{i=1}^{K} \log p(a_i^*|d_i,\mathcal{S};\theta_d),\end{aligned} \tag{16}$$

where d_i is computed by Eq. 11 with inputs of d_{i-1} and a_{i-1} now, here a_{i-1} is sampled from the trained policy $p(a_{i-1}|d_{i-1},\mathcal{S};\theta_d)$.

4.2.3 Learning from Rewards.

Reward Estimator. The objective of exact-K recommendation is to maximize the chance of being clicked or satisfied for the selected card A given candidate items set S and user u, as we defined in Sec. 3.1 and Eq. 1. Leveraging the advantage of reinforcement learning, we can directly optimize the objective by regarding it as **reward** function in RL. While there is no explicit reward in our problem, we can estimate the reward function based on teacher's demonstration by the idea from Inverse Reinforcement Learning [1]. The reward function can then be more generalized against supervised by demonstration only. In our problem, there are large amount of explicit feedback data in which users click cards (labeled as $r^*=1$) or not (labeled as $r^*=0$), we represent it as $P^D_{data}(r^*|A,u)$. Then we transfer estimation of reward function to the problem of CTR estimation for a card A given user u as $P(r=1|A,u;\phi)$, and the loss function for training it is as follows:

$$\begin{aligned}\mathcal{L}_D(\phi) = -\sum_{A,u,r^* \in P^D_{data}} \Big(& r^* \log\big(P(r=1|A,u;\phi)\big) + \\ & (1-r^*)\log\big(1-P(r=1|A,u;\phi)\big)\Big).\end{aligned} \tag{17}$$

To model the reward function, we follow the work of PNN [21] , in which we consider the feature crosses for card of items and user. And we define $P(r=1|A,u;\phi)$ as following equation:

$$\begin{aligned}&P(r=1|A,u;\phi) = \\ &\sigma\Big(W_{R2} ReLU\big(W_{R1}\big[[x_{a_i} \odot x_u]_{i=1}^K; [x_{a_i}]_{i=1}^K; x_u\big] + b_{R1}\big) + b_{R2}\Big),\end{aligned} \tag{18}$$

where $[\cdot]_{i=1}^K$ represents the concatenation of K vectors, $\odot$ is inner-product and σ means sigmoid function, x_{a_i} and x_u are feature vector for item a_i and user u defined in Sec. 4.1.1, $W_{R1}, W_{R2}, b_{R1}, b_{R2}$ are trainable parameters for reward function totally donated by ϕ.

REINFORCE with Hill-climbing. After we get the optimized reward function represented as $P(r=1|A,u;\phi^*)$, we use policy gradient based reinforcement learning (REINFORCE) [29] to train the policy. And its loss function given previously defined dataset $P^S_{data}(\cdot|S,u)$ is derived as follows:

$$\begin{aligned}\mathcal{L}_R(\theta) &= -\sum_{S,u \in P^S_{data}} \mathbb{E}_{A \sim P(A|S,u;\theta)}[R(A,u)] \\ &= -\sum_{S,u \in P^S_{data}} R(A,u) \sum_{i=1}^{K} \log p(a_i|a_1,\cdots,a_{i-1},\mathcal{S};\theta_d),\end{aligned} \tag{19}$$

where $\mathcal{S}$ is previously defined encoder state, $R(A,u)$ is the delayed reward [31] obtained after the whole card A is generated and is estimated by the following equation:

$$R(A,u) = 2 \times \big(P(r=1|A,u;\phi^*) - 0.5\big), \tag{20}$$

here we rescale the value of reward between -1.0 to 1.0.

One problem for training REINFORCE policy is that the reward is delayed and sparse, in which policy may be hard to receive positive reward signal, thus the training procedure of policy becomes unstable and falls into local minimal finally. In order to effectively avoid non-optimal local minimal and steadily increase the reward throughout training, we borrow the idea of Hill Climbing (HC) which is heuristic search used for mathematical optimization problems [8]. Instead of directly sampling from the policy by $A \sim P(A|S,u;\theta)$, in our method we first stochastically sample a buffer of $m=5$ solutions (cards) from policy and select the best one as A^*, then train the policy by A^* according to Eq. 19. In that case, we will always learn from a better solution to maximize reward, train on it and use the trained new policy to generate a better one.

4.2.4 Combination. To benefit from both fields of *Learning from Demonstrations* and *Learning from Rewards*, we simply apply linear combination of their loss functions and conduct the final loss as:

$$\mathcal{L}(\theta) = \alpha \times \mathcal{L}_S(\theta) + (1-\alpha) \times \mathcal{L}_R(\theta), \tag{21}$$

where $\mathcal{L}_S(\theta)$ and $\mathcal{L}_R(\theta)$ are formulated by Eq. 16 and 19, $\alpha \in [0, 1]$ is the hyper-parameter which should be tuned. The overall learning process is shown in Algorithm 2.

5 EXPERIMENTS

In this section, we conduct experiments with the aim of answering the following research questions:

- **RQ1** Does our proposed *GAttN with RLfD* method outperform the baseline methods in exact-K recommendation problem?
- **RQ2** How does our proposed *Graph Attention Networks* (GAttN) framework work for modeling the problem?
- **RQ3** How does our proposed optimization framework *Reinforcement Learning from Demonstrations* (RLfD) work for training the model?

5.1 Experimental Settings

5.1.1 Datasets. We experiment with three datasets and Tab. 3 summarizes the statistics. The first two datasets are constructed from MovieLens and last dataset is collected from real-world exact-K recommendation system on Taobao platform. The implementation details and parameter settings can be found in Appx. C.2.

MovieLens. This movie rating dataset[6] has been widely used to evaluate collaborative filtering algorithms in RS. As there is no public datasets to tackle exact-K recommendation problem, we construct for it based on MovieLens. While MovieLens is an explicit feedback data, we first transform it into implicit data, where we regard the 5-star ratings as positive feedback and treat all other entries as unknown feedback [29]. Then we construct recommended cards of each user with set of K items[7], where cards containing positive item are regarded as positive cards (labeled as 1) and cards without any positive item are negative cards (labeled as 0). Meanwhile, positive item in the corresponding card can be seen as what user actually clicked or preferred item belonging to that card. Finally we construct a candidate set with N items for each card for a user, where the N items are randomly sampled from the whole items set given this user and must include all the items in that card. We show examples how the dataset like in Tab. 4. Specially in our experiments, we construct two dataset: 1) card with $K = 4$ items along with $N = 20$ candidate items and 2) card with $K = 10$ items along with $N = 50$ candidate items. We call the above two dataset as **MovieLens(K=4,N=20)** and **MovieLens(K=10,N=50)**. Notice that there is no constraint between items in the output card (i.e $C = \emptyset$ defined in Sec. 3.1) for these two datasets.

Taobao. Above two datasets for exact-K recommendation problem based on MovieLens are what we call synthetic data which are not real-world datasets in production. On the contrary, we collect a dataset from exact-K recommendation system in Taobao platform, of which two days' samples are used for training and samples of the following day for testing, and specifically with $K = 4$ and $N = 50$. We call it **Taobao(K=4,N=50)**. Notice there is a required constraint between items in the output card in this dataset, that normalized edit distance (NED) [19] of any two items' titles must be larger than $\tau = 0.5$, i.e $C = \{NED(a_i, a_j) \geq \tau | a_i \in A, a_j \in A, i \neq j\}$ defined in Sec. 3.1, to guarantee the diversity of items in a card.

[6] http://grouplens.org/datasets/movielens/100k/

[7] The K items in a card are randomly permuted. As we suppose in Sec. 3.1, the permutation of the K item is not considered.

5.1.2 Evaluation Protocol. For evaluation, we can't use traditional ranking evaluation metrics such as nDCG, MAP, etc. These metrics either require prediction scores for individual items or assume that "better" items should appear in top ranking positions, which are not suitable for exact-K recommendation problem.

Hit Ratio. Hit Ratio (HR) is a recall-based metric, measuring how much the testing ground-truth K items of card A^* are in the predicted card A with exact K items. Specially for exact-K recommendation, we refer to HR@K and is formulated as follows:

$$HR@K = \sum_{i=1}^{n} \frac{|A_i \cap A_i^*|}{K} \Big/ n \,, \tag{22}$$

where n is the number of testing samples, $|\cdot|$ represents the number of items in a set.

Precision. Precision (P) measures whether the actually clicked (positive) item a^* in ground-truth card is also included in the predicted card A with exact K items, and is formulated as follows:

$$P@K = \sum_{i=1}^{n} I(a_i^* \in A_i) \Big/ n \,, \tag{23}$$

where $I(\cdot) \in \{0, 1\}$ is the indicator function.

5.1.3 Baselines. Our baseline methods are based on Naive Node-Weight Estimation Method (in Sec. 3.2) to adapt to exact-K recommendation. The center part is to estimate node weight which can be seen as CTR for the corresponding item. Therefor LTR based methods are applied and we compare with the follows:

Pointwise Model. DeepRank model is a popular ranking method in production which applies DNNs and a pointwise ranking loss (a.k.a MLE) [14].

Pairwise Model. BPR [22] is the method optimizes MF model [17] with a pairwise ranking loss. It is a highly competitive baseline for item recommendation.

Listwise Model. GRU based listwise model (Listwise-GRU) a.k.a DLCM [2] is a SOTA model for whole-page ranking refinement. It applies GRU to encode the candidate items with a list-wise ranking loss. In addition, we also compare with listwise model based on Multi-head Self-attention in Sec. 4.1.2 as Listwise-MHSA.

5.2 Performance Comparison (RQ1)

Tab. 1 shows the performances of P@K and HR@K for the three datasets with respect to different methods. First, we can see that our method with the best setting (*GAttN with RLfD*) achieves the best performances on both datasets, significantly outperforming the SOTA methods Listwise-MHSA and Listwise-GRU by a large margin (on average over three datasets, the relative improvements for P@K and HR@K are **7.7%** and **4.7%**, respectively). Secondly from the results, we can find that listwise methods (both Listwise-MHSA and Listwise-GRU) outperform pointwise and pariwise baselines

significantly. Therefore listwise methods are more suitable for exact-K recommendation, because they consider the context information to represent an item (node in graph) as what we have claimed in Sec. 4.1.2. And Listwise-MHSA performs better than Listwise-GRU, which indicates the effectiveness of our proposed MHSA method for encoding the candidate items (graph nodes). More detailed analysis for our method *GAttN with RLfD* can be found in the following two subsections (RQ2 and RQ3).

Table 1: Overall performances respect to different methods on three datasets, where $*$ means a statistically significant improvement for $p < 0.01$.

Model	MovieLens (K=4,N=20)		MovieLens (K=10,N=50)		Taobao (K=4,N=50)	
	P@4	HR@4	P@10	HR@10	P@4	HR@4
DeepRank	0.2120	0.1670	0.0854	0.1320	0.6857	0.6045
BPR	0.3040	0.2050	0.2350	0.1801	0.7357	0.6582
Listwise-GRU	0.4142	0.2423	0.4041	0.2144	0.7645	0.6942
Listwise-MHSA	0.4272	0.2465	0.4384	0.2168	0.7789	0.7176
Ours (best)	**0.4743**	**0.2611**	**0.4815**	**0.2245**	**0.7958**	**0.7488**
Impv.	11.0%*	6.1%*	9.8%*	3.6%	2.2%	4.3%

5.3 Analysis for GAttN (RQ2)

Tab. 1 shows that Listwise-MHSA performs better than Listwise-GRU on both P@K and HR@K on all datasets. It indicates the effectiveness to apply MHSA method for encoding the candidate items (graph nodes). As we claimed in Sec. 4.1.2, the representation for a node should consider the other nodes in graph, for there can exist some underlying structures in graph that nodes may influence between each other. Here we further give a presentational case on how the self-attention works in encoder (see Fig. 4) based on Taobao dataset. Take item "hat" in graph for example, items with larger attention weights to it are kinds of "scarf", "glove" and "hat". It is reasonable that users focusing on "hat" tend to prefer "scarf" rather than "umbrella". So to represent item "hat" in graph, it's helpful to attend more features of items like "scarf".

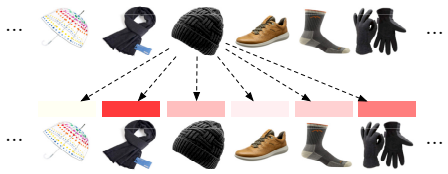


Figure 4: An example of the attention mechanism in the encoder self-attention in layer 2. The higher attention weights of the item, the darker color of the grid. We take item "hat" for example and only show attention weights in one head.

Beam search is proposed in GAttN decoder (see Sec. 4.1.2) to expand the search space and try more combinations of items in a card to get a most optimal solution. A critical hyper-parameter for beam search is the beam size, indicating how many solutions to search in a decoding time-step. We tune beam size in Fig. 5 and find that larger beam size can lead to better performances[8] on both P@K and HR@K. However we can also see that when beam size gets larger than 3 the improvement of performances will be minor, so for efficiency consideration we set beam size as 3 in our experiments.

[8]We only report the results on MovieLens(K=4,N=20) and other datasets follow the same conclusion.

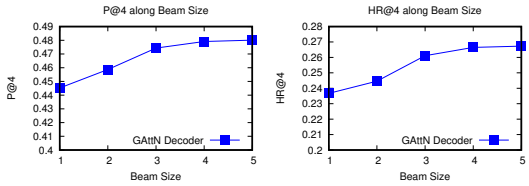


Figure 5: Performance of P@4 and HR@4 with different beam size in GAttN decoder.

5.4 Analysis for RLfD (RQ3)

To verify how our proposed optimization framework *RLfD* works, we will do the following ablation tests:

(1) Set $\alpha = 0$ in Eq. 21 (only reinforcement loss as Eq. 19), and compare *RL(w/ hill-climbing)* with *RL(w/o hill-climbing)*.
(2) Set $\alpha = 1$ in Eq. 21 (only supervised loss as Eq. 16), and compare *SL(w/ policy-sampling)* with *SL(w/o policy-sampling)*.
(3) Finetune α in Eq. 21 and figure out the influence to the combination of SL with RL.

Here we represent *Learning from Demonstrations* and *Learning from Rewards* as SL and RL for short. Tab. 2 gives an overall results and we only report on dataset MovieLens(K=4,N=20) for simplification.

Table 2: Performance for different settings in RLfD.

	Settings in RLfD	MovieLens (K=4,N=20)	
		P@4	HR@4
1	RL(w/o hill-climbing)	0.3340	0.2314
2	RL(w/ hill-climbing)	0.3573	0.2330
3	SL(w/o policy-sampling)	0.4095	0.2401
4	SL(w/ policy-sampling)	0.4272	0.2465
5	RL(w/o hill-climbing) + SL(w/ policy-sampling)	0.4495	0.2514
6	RL(w/ hill-climbing) + SL(w/o policy-sampling)	0.4472	0.2534
7	**RL(w/ hill-climbing) + SL(w/ policy-sampling)**	**0.4743**	**0.2611**

Fig. 6 shows the learning curves respect to Reward (defined in Eq. 20), P@4 and HR@4 for RL with (w/) or without (w/o) hill-climbing proposed in Sec. 4.2.3. From the curves, we can find that with the help of hill-climbing REINFORCE training becomes more stable and steadily improves the performance, finally achieves a better solution (row 1 vs. 2 and row 5 vs. 7 in Tab. 2). Another insight in Fig. 6 is that learning curve of Reward is synchronous monotonous with P@4 and HR@4, which verifies the effectiveness of our defined reward function to direct the objective in problem.

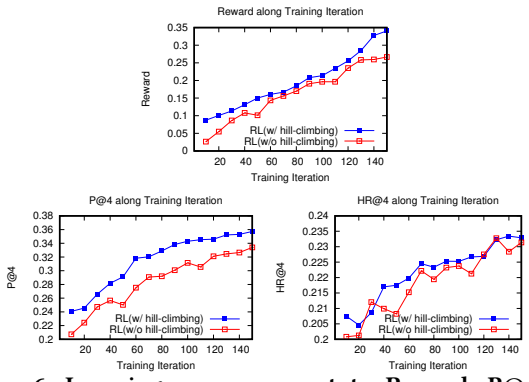


Figure 6: Learning curves respect to Reward, P@4 and HR@4 for RL with (w/) or without (w/o) hill-climbing.

Fig. 7 shows the learning curves respect to P@4 and HR@4 for SL with (w/) or without (w/o) policy-sampling proposed in Sec. 4.2.2. We observe that in the beginning 50 iterations SL with policy-sampling may perform worse than without policy-sampling. We believe that in the first steps of training procedure the learned policy can be poor, so feeding the output sampled from such policy to the next time-step as input in decoder can lead to worse performances. However as training goes on, SL with policy-sampling will converge better for revising the inconsistency between training and inference of policy, finally achieve better performances (row 3 vs. 4 and row 6 vs. 7 in Tab. 2).

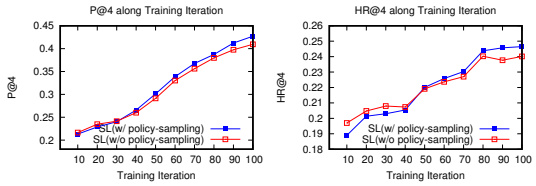


Figure 7: Learning curves respect to P@4 and HR@4 for SL with (w/) or without (w/o) policy-sampling.

In Fig. 8 we tune hyper-parameter α defined in Eq. 21, which represents trade-off for applying SL and RL in training process. We observe that $\alpha = 0.5$ achieves the best both on P@4 and HR@4. The performances increase when α is tuned from 0 to the optimal value and then drops down afterwards, which indicates that properly combining SL and RL losses can result in the best solution. Furthermore, we find that when only apply SL loss ($\alpha = 1$) we can get a preliminary sub-optimal policy, after involving some degree of RL loss the policy can be directed to achieve more optimal solutions, which verifies the sufficiency-and-efficiency of our proposed *Reinforcement Learning from Demonstrations* to train the policy.

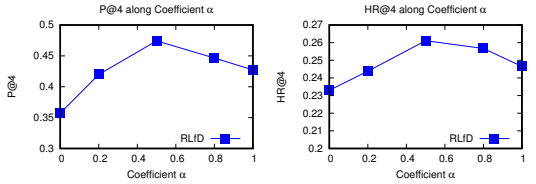


Figure 8: Performance of P@4 and HR@4 with different coefficients α in loss defined in Eq. 21.

6 CONCLUSION AND FUTURE WORK

This work targets to a practical recommendation problem named exact-K recommendation, we prove that it is different from traditional top-K recommendation. In the first step, we give a formal problem definition, then reduce it to a Maximal Clique Optimization problem which is a combinatorial optimization problem and NP-hard. To tackle this specific problem, we propose a novel approach of *GAttN with RLfD*. In our evaluation, we perform extensive analysis to demonstrate the highly positive effect of our proposed method targeting exact-K recommendation problem. In our future work, we plan to adopt adversarial training for the components of Reward Estimator and REINFORCE learning, regarding as discriminator and generator in GAN's [29] perspective. Moreover further online A/B testing in production will be conducted.

REFERENCES

[1] Pieter Abbeel and Andrew Y Ng. 2004. Apprenticeship learning via inverse reinforcement learning. In *ICML*.
[2] Qingyao Ai, Keping Bi, Jiafeng Guo, and W Bruce Croft. 2018. Learning a Deep Listwise Context Model for Ranking Refinement. In *SIGIR*.
[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. In *arXiv*.
[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *arXiv*.
[5] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. 2016. Neural Combinatorial Optimization with Reinforcement Learning. In *arXiv*.
[6] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *NIPS*.
[7] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*.
[8] Lu Duan, Haoyuan Hu, Yu Qian, Yu Gong, Xiaodong Zhang, Yinghui Xu, and Jiangwen Wei. 2019. A Multi-task Selected Learning Approach for Solving 3D Flexible Bin Packing Problem. In *AAMAS*.
[9] Michael R Garey, David S Johnson, and Larry Stockmeyer. 1974. Some simplified NP-complete problems. In *STOC*.
[10] Yu Gong, Xusheng Luo, Kenny Q Zhu, Wenwu Ou, Zhao Li, and Lu Duan. 2018. Automatic generation of chinese short product titles for mobile display. In *arXiv*.
[11] Yu Gong, Xusheng Luo, Yu Zhu, Wenwu Ou, Zhao Li, Muhua Zhu, Kenny Q Zhu, Lu Duan, and Xi Chen. 2018. Deep Cascade Multi-task Learning for Slot Filling in Online Shopping Assistant. In *arXiv*.
[12] Yu Gong, Kaiqi Zhao, and Kenny Qili Zhu. 2016. Representing verbs as argument concepts. In *AAAI*.
[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.
[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*.
[15] Adrian Ion, JoÃčo Carreira, and Cristian Sminchisescu. 2011. Image segmentation by figure-ground composition into maximal cliques. In *ICCV*.
[16] Ray Jiang, Sven Gowal, Yuqiu Qian, Timothy Mann, and Danilo J Rezende. 2018. Beyond Greedy Ranking: Slate Optimization via List-CVAE. In *arXiv*.
[17] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* (2009).
[18] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *arXiv*.
[19] Andres Marzal and Enrique Vidal. 1993. Computation of normalized edit distance and applications. *PAMI* (1993).
[20] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. 2018. Overcoming exploration in reinforcement learning with demonstrations. In *ICRA*.
[21] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *ICDM*.
[22] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*.
[23] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.
[24] Faraz Torabi, Garrett Warnell, and Peter Stone. 2018. Behavioral Cloning from Observation. In *arXiv*.
[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.
[26] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2015. Order matters: Sequence to sequence for sets. In *arXiv*.
[27] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *NIPS*.
[28] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *CVPR*.
[29] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR*.
[30] Yunlun Yang, Yu Gong, and Xi Chen. 2018. Query Tracking for E-commerce Conversational Search: A Machine Comprehension Perspective. In *CIKM*.
[31] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient.. In *AAAI*.
[32] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A Deep Reinforcement Learning Framework for News Recommendation. In *WWW*.
[33] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. 2017. What to Do Next: Modeling User Behaviors by Time-LSTM.. In *IJCAI*.
[34] Yu Zhu, Junxiong Zhu, Jie Hou, Yongliang Li, Beidou Wang, Ziyu Guan, and Deng Cai. 2018. A brand-level ranking system with the customized attention-GRU model. In *IJCAI*.

A NAIVE NODE-WEIGHT ESTIMATION METHOD

Algorithm 1 Naive Node-Weight Estimation Method.

Require: Given user u and candidate items set S, construct graph $\mathbb{G}(\mathcal{N}, \mathcal{E})$ defined in paragraph 2 of Sec. 3.1.

Estimate weight w_i of each node $n_i \in \mathcal{N}$ in graph based on CTR of corresponding item $s_i \in S$.
Initial result card $A = \emptyset$.
for *t = 1 to K* **do**
    Select node a_t with the largest weight in $\mathcal{N}$ and add to A.
    Remove a_t and nodes in $\mathcal{N}$ which are not adjacent to a_t.
end for
return result card A.

B REINFORCEMENT LEARNING FROM DEMONSTRATIONS

Algorithm 2 Reinforcement Learning from Demonstrations.

Phase 1 - Reward Estimator Training

Require: reward function $P(r = 1|A, u; \phi)$, dataset $P^D_{data}(r^*|A, u)$

Optimize ϕ with gradient descent by loss function $\mathcal{L}_D(\phi)$.
return $P(r = 1|A, u; \phi^*)$

Phase 2 - Policy Training

Require: optimized reward function $P(r = 1|A, u; \phi^*)$, dataset $P^S_{data}(A^*|S, u)$, policy $P(A|S, u; \theta)$

Optimize θ with gradient descent by loss function $\mathcal{L}(\theta)$.
return $P(A|S, u; \theta^*)$

C EXPERIMENTAL SETTINGS

C.1 Datasets

Table 3: Statistics of the experimented datasets.

Dataset	User#	Card#	Item#	Sample#
MovieLens(K=4,N=20)	817	40036	1630	40036
MovieLens(K=10,N=50)	485	33196	1649	33198
Taobao(K=4,N=50)	581055	310509	3148550	1116582

Table 4: Show case of the dataset.

	user	card	candidate items	card label	positive item
sample#1	1	1,2,3,4	1,2,3,4,...,20	1	2
sample#2	1	1,4,5,6	1,2,3,4,...,20	0	/
			...		

(We take $K = 4$ and $N = 20$ for example. Items and users are represented as IDs here. Card label represents whether the card is clicked or satisfied by user (labeled as 1) or not (labeled as 0). Positive item is the actually clicked item in card by user.)

C.2 Implementation and Parameter Settings

Here we report implementation details for the three datasets[9] (two MovieLens based datasets and one Taobao based dataset), and our implementation is based on TensorFlow[10]. To construct the training and test sets, we perform a 4:1 random splitting as in [29] for all the datasets.

C.2.1 MovieLens. Notice both MovieLens(K=4,N=20) and MovieLens(K=10,N=50) share the same parameter settings. For a fair comparison, all models are set with an embedding size of 16 for item and user IDs, and optimized using the mini-batch Adam optimizer with a batch size of 32 and learning rate of 0.001. All models are trained for 10 epoch. All the trainable feed-forward parameter matrices are set with the same input and output dimension as 32×32 (including DeepRank, BPR, and all the RNN cells in both Listwise-GRU, Listwise-MHSA and ours). Specifically for our GAttN model, in decoder (in Sec. 4.1.3) we use LSTM cells with units number of 32 and set beam size as 3, number of heads in encoder (in Sec. 4.1.2) MHSA layer is 2, and the coefficient parameter α in loss function (in Sec. 4.2.4) is 0.5. Number of layers L in both encoder and decoder are set as 2. For reward estimator model (in Sec. 4.2.3), we set the hidden size in fully-connected layer as 128.

C.2.2 Taobao. In this dataset, the feature vectors for user and item are statistic features with size of 40 and 52 specifically, instead of ID features. Sample statistic features are PV (page view), IPV (item page view), GMV (cross merchandise volume), CTR (click through rate) and CVR (conversion rate) for 1 day, 7 days and 14 days, etc. For this dataset, we first transfer the input representation of user and item to 32 dimension, i.e we set $W_I \in \mathbb{R}^{92\times32}$ and $b_I \in \mathbb{R}^{32}$ in Sec. 4.1.1. And all the other hyper-parameters are set as the same with those on MovieLens based datasets (refer to Appx. C.2.1).

[9] https://github.com/pangolulu/exact-k-recommendation
[10] https://www.tensorflow.org/