Unified and Scalable Incremental Recommenders with Consumed Item Packs

Rachid Guerraoui[1], Erwan Le Merrer[2(✉)], Rhicheek Patra[3], and Jean-Ronan Vigouroux[4]

[1] EPFL, Lausanne, Switzerland
Rachid.Guerraoui@epfl.ch
[2] Univ Rennes, Inria, CNRS, IRISA, Rennes, France
erwan.le-merrer@inria.fr
[3] Oracle Labs, Zurich, Switzerland
rhicheek.patra@oracle.com
[4] Technicolor, Rennes, France
jean-ronan.vigouroux@technicolor.com

Abstract. Recommenders personalize the web content using collaborative filtering to relate users (or items). This work proposes to unify *user-based*, *item-based* and *neural word embeddings* types of recommenders under a single abstraction for their input, we name Consumed Item Packs (CIPs). In addition to genericity, we show this abstraction to be compatible with incremental processing, which is at the core of low latency recommendation to users. We propose three such algorithms using CIPs, analyze them, and describe their implementation and scalability for the Spark platform. We demonstrate that all three provide a recommendation quality that is competitive with three algorithms from the state-of-the-art.

Keywords: Implicit recommenders · Incremental updates · Parallelism · Spark

1 Introduction

Recent recommender systems exploit implicit feedback [1–3] (*i.e.*, they do not leverage *ratings* collected from users), and show competitive results with Singular Value Decomposition (SVD) based recommenders [4]. They aim at uncovering high-order relations between consumed items. Each paper proposes a specific algorithm, with an arbitrary definition of sequences of consumed items. Our motivation is to investigate the existence of a higher level abstraction for sequences of consumed items, and algorithms for dealing with it. Such an abstraction, we name a *Consumed Item Pack* (CIP), allows to reason about and to propose sequence-aware algorithms within the same framework, capable of addressing implicit recommendation.

The challenges are threefold. *(i)* We first have to highlight that the notion of CIP captures the analogous consumption pattern of users (*e.g.*, the one exposed

R. Yahyapour (Ed.): Euro-Par 2019, LNCS 11725, pp. 227–240, 2019.
https://doi.org/10.1007/978-3-030-29400-7_17

in [1]). *(ii)* The second challenge is the computational complexity of the proposed algorithms in the CIP framework. Leveraging CIPs for building implicit recommenders is not immediate, for the computation time can easily become prohibitive given the size of user consumption logs in production systems. This is for instance the case in the previously introduced sequential approach HOSLIM [1], where algorithmic tractability is at stake. Section 2 presents three CIP based algorithms. Concerning memory-based Collaborative Filtering (CF), we show in Subsect. 2.1 (resp. Subsect. 2.2) how to build a CIP based similarity metric that is *incremental*, which helps in designing an implicit user-based (resp. item-based) recommender that *scales* while providing good recommendation quality. Moreover, we also present a model-based CF technique incorporating CIPs in Subsect. 2.3, which leverages neural word embeddings [5]. We demonstrate that our techniques scale with an increasing number of computing nodes while achieving a speedup comparable to Spark's Alternating Least Squares (ALS) recommender from the MLLIB library. *(iii)* These proposed implicit algorithms have to provide an accuracy that is at least comparable with classic CF recommenders, in order to be adopted in practice. For assessing their performance, we then conduct a comparison with an explicit SVD-based recommender [4], with an implicit one [6], as well as with a recent state-of-the-art algorithm [7] incorporating both implicit and explicit techniques.

Consumed Item Packs. Our CIPs relate to high order relations between items enjoyed by a user. Some previous works such as HOSLIM [1], considered the consumption of items by the same user as the basis for implicit recommendation. HOSLIM places the so called *user-itemsets* (implicit feedback) in a matrix, and then computes the similarity of jointly consumed items over the whole user history (that leads to the optimal recommendation quality). High-order relations are sought in principle, but due to the tractability issue of this approach (for m items and order k: $O(m^k)$ combinations of the items are enumerated and tested for relevance), authors limit computations only to pairs of items. Recently, Barkan et al. proposed to consider item-item relations using the model of word embeddings in their technical report [2]. Our work generalizes the notion of implicit item relations, based on consumption patterns.

To get access to useful information from service logs, we define the CIP data structure. CIPs are extracted from users' consumption patterns, and allow us to compute the similarity between those users (or items consumed by them). A user's profile is composed of multiple CIPs. The notion of CIP is then instantiated in three different algorithms: in a user-based algorithm (Subsect. 2.1), in an item-based one (Subsect. 2.2) and in a word embedding based one (Subsect. 2.3).

To make things more precise, consider a set of m users $\mathcal{U} = \{u_1, u_2, ..., u_m\}$ and a set of n items from a product catalog $\mathcal{I} = \{i_1, i_2, ..., i_n\}$. The transaction history of a user u, consists of a set of pairs of the form $\langle i, t_{ui} \rangle$ (where u consumed an item i at a time $t_{u,i}$), extracted from service logs. We denote u's profile as P_u, which consists of the time-ordered items in the log. CIPs are composed of items: each CIP $\in \mathcal{I}^*$. The order of the items in a given user's CIP represents

their relative appearance in time, the leftmost symbol being the oldest one: $\text{CIP}_u = [i_1, i_2, i_3, ..., i_k]$ *such that* $t_{u,i_1} < t_{u,i_2} < ... < t_{u,i_k}$.

A CIP then represents the items consumed by a user over a predefined period of time. Using such a data structure, one can devise a *similarity* measure $sim : \mathcal{I}^* \times \mathcal{I}^* \rightarrow \mathbb{R}^+$ between two CIPs, that captures the proximity between users (or items) as we explain it in the next section.

2 CIP Based Algorithms

The core claim of this paper is that the notion of CIP is general enough to capture different types of algorithms that rely on sequences of items. In the next three subsections, we present novel algorithms that determine CIP based similarities and leverage sequence of items for recommendation. To illustrate the generality of CIPs, the last subsection shows how a previously introduced algorithm (FISM [3]) is captured by the CIP framework.

2.1 CIP-U: A User-Based Recommender

CIP-U is an incremental algorithm that maintains a user-user network where each user is connected to the most similar K other users. CIP-U exploits users' CIPs, and accepts batches of items freshly consumed by users (*i.e.*, last logged transactions on the service) to update this network.

P_u^l denotes the profile of a user u till the l^{th} update of her consumed items, while CIP_u^{l+1} denotes the batch of new items consumed by her since the last batch update. Assuming $P_u^l = i_1 i_2 ... i_k$ and $\text{CIP}_u^{l+1} = i_{k+1} i_{k+2} ... i_n$, we can denote the profile of a user u after the $(l+1)^{th}$ iteration as $P_u^{l+1} = P_u^l \cup \text{CIP}_u^{l+1}$. Note that $\cup$ is an order preserving union here.

Before we provide the similarity measure to compare users, we introduce some preliminary definitions. We first introduce the notion of *hammock distance* between a pair of items in the profile of a given user u.

Definition 1 (Hammock distance). *The hammock distance between a pair of items (i, j) in P_u, denoted by $\mathcal{H}_u(i, j)$, is the number of hops between them.*

For instance, in $P_u = [i_{14}, i_3, i_{20}, i_{99}, i_{53}, i_{10}, i_{25}]$, $\mathcal{H}_u(i_{14}, i_{99}) = 3$.

Definition 2 (Hammock pairs). *Given two users u and v, their hammock pairs $\mathcal{HP}_{u,v}$ are the set of distinct item pairs both present in P_u and in P_v, under the constraint that the number of hops between pairs is at most δ_H.*

$$\mathcal{HP}_{u,v} = \{(i, j) \mid \mathcal{H}_u(i, j) \leq \delta_H \quad \wedge \quad \mathcal{H}_v(i, j) \leq \delta_H \quad \wedge \quad i \neq j\}$$

Hyper-parameter δ_H denotes the *hammock threshold* and serves the purpose of tuning the CIP based latent feature considered between related items.

Let $[\,]$ denote the Iverson bracket: $[\text{P}] = 1$ if P is True, 0 otherwise. From hammock pairs, we derive the similarity of two users with regards to their CIPs:

Definition 3 (Similarity measure for user-based CIP). *The similarity between two users u and v is defined as a function of the cardinality of the set of hammock pairs between them:*

$$sim_{\text{CIP-U}}(u,v) = 1 - (1 - [P_u = P_v]) \cdot e^{-|\mathcal{HP}_{u,v}|} \quad (1)$$

We obtain $sim_{\text{CIP-U}} \in [0,1]$, with the boundary conditions, $sim_{\text{CIP-U}} = 0$ if the two users have no pair in common ($|\mathcal{HP}_{u,v}| = 0$ and $[P_u = P_v] = 0$), while $sim_{\text{CIP-U}} = 1$ if their CIPs are identical ($[P_u = P_v] = 1$).

Incremental Updates. CIP-U enables incremental updates, in order to conveniently reflect the latest users' consumption in recommendations without requiring a prohibitive computation time. CIP-U processes batches of events (consumed items) at regular intervals and updates the similarity measure for pairs of users. $C_{u,v}$ denotes the set of items common in the profiles of two users u and v. More precisely, after the l^{th} iteration, we obtain: $C^l_{u,v} = P^l_u \cap P^l_v$. Then, at the $(l+1)^{th}$ iteration, we get:

$C^{l+1}_{u,v} = P^{l+1}_u \cap P^{l+1}_v = (P^l_u \cup \text{CIP}^{l+1}_u) \cap (P^l_v \cup \text{CIP}^{l+1}_v) = (P^l_u \cap P^l_v) \cup (P^l_u \cap \text{CIP}^{l+1}_v) \cup (P^l_v \cap \text{CIP}^{l+1}_u) \cup (\text{CIP}^{l+1}_u \cap \text{CIP}^{l+1}_v) = C^l_{u,v} \cup \Delta C^{l+1}_{u,v}$, where $\Delta C^{l+1}_{u,v} = (P^l_u \cap \text{CIP}^{l+1}_v) \cup (P^l_v \cap \text{CIP}^{l+1}_u) \cup (\text{CIP}^{l+1}_u \cap \text{CIP}^{l+1}_v)$. Note that the time complexity of this step is $O((|P^l_u| + |\text{CIP}^{l+1}_v|) + (|P^l_v| + |\text{CIP}^{l+1}_u|))$, where $|\text{CIP}^{l+1}_u|$, $|\text{CIP}^{l+1}_v|$ are bounded by the number of events, say Q, after which the batch update will take place. Hence, the time complexity is $O(n+Q) = O(n)$, where n denotes the total number of items, and when Q is a constant (and $Q << n$ as expected in a system built for incremental computation).

We next incrementally compute the new hammock pairs. $\Delta\mathcal{HP}_{u,v}$ denotes the set of new hammock pairs for users u and v. Computation is performed as follows: $\Delta\mathcal{HP}_{u,v} = \{(i,j) \mid (i \in C^l_{u,v}, j \in \Delta C^{l+1}_{u,v}) \ \wedge \ (i \in \Delta C^{l+1}_{u,v}, j \in \Delta C^{l+1}_{u,v}) \ \wedge \ \mathcal{H}_u(i,j) \leq \delta_H \ \wedge \ \mathcal{H}_v(i,j) \leq \delta_H\}$.

The time complexity of this step is $O(|C^l_{u,v}| \cdot |\Delta C^{l+1}_{u,v}|)$, where $|\Delta C^{l+1}_{u,v}|$ is bounded by the number of events after which the batch update takes place (Q). Hence, the time complexity is also of $O(n \cdot Q) = O(n)$.

Finally, the similarities are computed leveraging the cardinality of the computed incremental hammock pairs. More precisely, we compute the updated similarity on-the-fly between a pair of users u and v after the $(l+1)^{th}$ iteration as follows: $sim^{l+1}_{u,v} = 1 - (1 - [P^{l+1}_u = P^{l+1}_v]) \cdot e^{-|\mathcal{HP}^l_{u,v} + \Delta\mathcal{HP}_{u,v}|}$.

Hence, the similarity computation between one user and all m others is $O(nm)$. In CIP-U, we retain a small number K of the most similar users (where $K << m$) per given user. Selecting the top-K similar users for collaborative filtering based on their similarity requires sorting, which induces an additional $O(m \log m)$. The total complexity is $O(nm) + O(m \log m) = O(nm)$ (since $n >> \log m$). Note that classic explicit collaborative filtering algorithms (user or item-based) have same time complexity for periodically updating their recommendation models. Note that complexity for the top-K neighbors can be reduced further to $O(n)$ by using biased sampling and iteratively updating neighbors [8].

2.2 CIP-I: An Item-Based Recommender

CIP-I is also an incremental algorithm that processes user consumption events in CIPs, to update its item-item network. Similar to CIP-U, we also leverage the notion of user *profiles*: a profile of a user u is noted P_u, and is composed of one or more disjoint CIPs. We use multiple CIPs in a user profile to model her consumption pattern. CIPs are separated based on the timestamps associated with the consumed items: two consecutive CIPs are disjoint if the former's last and latter's first items are separated in time by a given interval δ.

Definition 4 (CIP PARTITIONS IN A USER PROFILE). *Let i_k and i_{k+1} denote two consecutive consumption events of a user u, with consumption timestamps t_{u,i_k} and $t_{u,i_{k+1}}$, such that $t_{u,i_k} \leq t_{u,i_{k+1}}$. Given i_k belongs to CIP_u^l, item i_{k+1} is added to CIP_u^l if $t_{u,i_{k+1}} \leq t_{u,i_k} + \delta$. Otherwise i_{k+1} is added as the first element in a new CIP_u^{l+1}.*

These CIPs are defined as δ-distant. The rationale behind the creation of user profiles composed of CIPs is that each CIP is intended to capture the semantic taste of a user within a consistent consumption period.

With $i <_{\text{CIP}} j$ denoting the prior occurrence of i before j in a given CIP, and the inverse hammock distance $\epsilon_u(i,j)$ being a penalty function for distant items in a CIP_u (*e.g.*, $\epsilon_u(i,j) = \frac{1}{\mathcal{H}_u(i,j)}$), we express a similarity measure for items, based on those partitioned user profiles, as follows.

Definition 5 (SIMILARITY MEASURE FOR ITEM-BASED CIP). *Given a pair of items (i,j), their similarity is:*

$$sim_{\text{CIP-I}}(i,j) = \frac{\sum_u \sum_{l=1}^{|l|_u} [(i,j) \in \text{CIP}_u^l \wedge i <_{\text{CIP}} j](1 + \epsilon_u(i,j))}{2 \cdot \max\{\sum_u \sum_{l=1}^{|l|_u} [i \in \text{CIP}_u^l], \sum_u \sum_{l=1}^{|l|_u} [j \in \text{CIP}_u^l]\}} = \frac{score_{\text{CIP-I}}(i,j)}{2 \cdot max\{cardV(i), cardV(j)\}}, \quad (2)$$

with $|l|_u$ the number of CIP*s in u's profile, and $[\,]$ the Iverson bracket.*

This reflects the number of close and ordered co-occurrences of items i and j over the total number of occurrences of both items independently: $sim_{\text{CIP-I}}(i,j) = 1$ if each appearance of i is immediately followed by j in the current CIP. Contrarily, $sim_{\text{CIP-I}}(i,j) = 0$ if there is no co-occurrence of those items in any CIP. Furthermore, we denote the numerator term as $score_{\text{CIP-I}}(i,j)$ and the denominator term as a function of $cardV(i)$ and $cardV(j)$ sub-terms for Eq. 2, where $cardV(i) = \sum_u \sum_{l=1}^{|l|_u} [i \in \text{CIP}_u^l]$. As shown in Algorithm 1, we can update $score_{\text{CIP-I}}(i,j)$ and $cardV(i)$ terms incrementally. Finally, we compute the similarity on-the-fly with the $score_{\text{CIP-I}}(i,j)$ and $cardV(i)$ terms.

Incremental Updates. CIP-I processes users' recent CIPs scanned from users' consumption logs. Score values ($score_{\text{CIP-I}}$) are updated (Algorithm 1). We

require an item-item matrix to maintain the *score* values, as well as a n-dimensional vector that maintains the current number of occurrences of each item.

After the update of the *score* values, the algorithm terminates by updating a data structure containing the top-K closest items for each given item, leveraging the *score* matrix and the cardinality terms for computing the similarities on-the-fly.

Algorithm 1. *Incremental Updates for Item Pairs.*

Require: CIP_u ▷ last δ-distant CIP received for user u
1: $score_{\text{CIP-I}}[][]$ ▷ item-item *score* matrix, initialized to 0
2: $cardV$ ▷n-dim. vector of appearance cardinality of items
3: **for** item i in CIP_u **do**
4: $cardV(i) = cardV(i) + 1$
5: **for** item j in CIP_u **do**
6: **if** $i \neq j$ **then**
7: $\epsilon(i,j) = \epsilon(j,i) = \frac{1}{\mathcal{H}_u(i,j)}$
8: **if** $i <_{\text{CIP}} j$ **then**
9: $score_{\text{CIP-I}}[i][j] \mathrel{+}= (1 + \epsilon(i,j))$
10: **else**
11: $score_{\text{CIP-I}}[j][i] \mathrel{+}= (1 + \epsilon(j,i))$

The complexity of Algorithm 1 depends on the maximum tolerated size of incoming CIPs. As one expects an incremental algorithm to receive relatively small inputs as compared to the total dataset size, the final complexity is compatible with online computation: *e.g.*, if the largest CIP allowed has cardinality $|\text{CIP}| = O(\log n)$, then run-time complexity is poly-logarithmic.

2.3 DeepCIP: An Embedding-Based Recommender

In this subsection, we present an approach based on machine learning, inspired by Word2Vec [2,5]. This approach relies on word embedding, transposed to items. We specifically adapt this concept to our CIP data structure.

Neural word embeddings, introduced in [5,9], are learned vector representations for each word from a text corpus. These neural word embeddings are useful for predicting the surrounding words in a sentence. A common approach is to use a multi-layer Skip-gram model with negative sampling. The objective function minimizes the distance of each word with its surrounding words within a sentence while maximizing the distances to randomly chosen set of words (*negative samples*) that are not expected to be close to the target. This is an objective quite similar to ours as it enables to compute proximity between items in the same CIP. With DeepCIP, we feed a Skip-gram model with item-pairs in CIPs where each CIP is as usual an ordered set of items (similar to the instantiation in CIP-I). More precisely, CIPs are δ-distant as instantiated in Subsect. 2.2.

DeepCIP trains the neural network with pairs of items at a distance less than a given *window size* within a CIP. This window size corresponds to the notion of hammock distance (defined in Subsect. 2.1) where the distance hyper-parameter δ_H is defined by the *window size*. More formally, given a sequence of T training items' vectors $i_1, i_2, i_3, ..., i_T$, and a maximum hammock distance of k, the objective of the DeepCIP model is to maximize the average log probability:

$$\frac{1}{T}\sum_{t=k}^{T-k} log\ P(i_t|i_{t-k}, ..., i_{t-1}, i_{t+1}, ..., i_{t+k}). \tag{3}$$

The Skip-gram model is employed to solve the optimization objective 3, where the weights of the model are learned using back-propagation and stochastic gradient descent. We implement DeepCIP using asynchronous stochastic gradient descent (Downpour-SGD [10]). Downpour-SGD enables distributed training for the Skip-gram model on multiple machines by leveraging asynchronous updates from them. We use a publicly-available deep learning framework [11] which implements Downpour-SGD in a distributed setting. More precisely, DeepCIP trains the model using Downpour-SGD on the recent CIPs thereby updating the model incrementally.

DeepCIP uses a *most_ similar* functionality to select items to recommend to a user, using as input recently consumed items (the current CIP). We compute a CIP vector using the items in the given CIP and then use this vector to find most similar other items. More precisely, the *most_ similar* method uses the cosine similarity between a simple mean of the projection weight vectors of the recently consumed items (i.e., items in a user's most recent CIP) and the vectors for each item in the database.

Incremental Updates. Online machine learning is performed to update a model when data becomes available. The DeepCIP model training is performed in an online manner [12], in which the model is updated using the recent CIPs. Online machine learning is crucial for recommendation systems, as it is necessary for the algorithm to dynamically adapt to new temporal patterns [13] in the data. Hence, the complexity of the model update is dependent on the number of new CIPs received along with the hyper-parameters for the learning algorithm (primarily: the Skip-gram model parameters, the dimensionality of item vectors, the number of training iterations, and the hammock distance).

2.4 The FISM Algorithm Under CIPs

We now demonstrate that the CIP framework can incorporate the state-of-art sequence-based algorithm FISM [3] (standing for Factored Item Similarity Models), in order to illustrate the generality of the CIP notion. In FISM, the item-item similarity is computed as a product of two low-ranked matrices $\mathbf{P} \in \mathcal{R}^{m\times k}$ and $\mathbf{Q} \in \mathcal{R}^{m\times k}$ where $k << m$. More precisely, the item-item similarity between any two items is defined as $sim(i, j) = \mathbf{p}_j \mathbf{q}_i^T$ where $\mathbf{p}_j \in \mathbf{P}$ and $\mathbf{q}_i \in \mathbf{Q}$.

Finally, the recommendation score for a user u on an unrated item i (denoted by $\bar{r}_{ui}$) is calculated as an aggregation of the items that have been rated by u:

$$\bar{r}_{ui} = b_u + b_i + (n_u^+)^{-\alpha} \sum_{j \in \mathcal{R}_u^+} \mathbf{p}_j \mathbf{q}_i^T, \tag{4}$$

where $\mathcal{R}_u^+$ is the set of items rated by user u (note that FISM do not leverage ratings, but only the fact that a rated item has been consumed by definition), b_u and b_i are the user and item biases, $\mathbf{p}_j$ and $\mathbf{q}_i$ are the learnt item latent factors, n_u^+ is the number of items rated by u, and α is a user specified parameter between 0 and 1. Moreover, term $(n_u^+)^{-\alpha}$ in Eq. 4 is used to control the degree of agreement between the items rated by the user with respect to their similarity to the item whose rating is being estimated (*i.e.*, item i).

We now present how Eq. 4 is adapted to fit into the CIP notion. For a user u, her profile (P_u) consists of $|l|_u$ different CIPs (similar to the notations introduced for Eq. 4). Equation 4 is rewritten with CIPs as:

$$\bar{r}_{ui} = b_u + b_i + (|\cup_{k=1}^{|l|_u} \mathrm{CIP}_u^k|)^{-\alpha} \sum_{k=1}^{|l|_u} \sum_{j \in \mathrm{CIP}_u^k} \mathbf{p}_j \mathbf{q}_i^T, \tag{5}$$

where $|\cdot|$ denotes the cardinality. We substitute consumed items by CIP structures; this last transformation shows that indeed CIPs incorporates the FISM definition of item sequences. We also note that due to the CIPs, the terms in Eq. 5 could be incrementally updated, similarly to CIP-U and CIP-I, by incorporating the latest CIP of user u.

3 Implementation with Spark and Evaluation

We first note that we open sourced our algorithms on GitHub [14]. We consider Apache Spark [15] as our framework for the computation of recommendations. Spark is a cluster computing framework for large-scale data processing; it provides several core abstractions, namely Resilient Distributed Datasets (RDDs), parallel operations and shared variables. We now introduce the RDDs adapted to our CIP-based algorithms.

RDDs for CIP-U. We store the collected information into three primary RDDs as follows. UsersRDD stores the information about the user profiles. UserSimRDD stores the hammock pairs between all pairs of users. The pairwise user similarities are computed using a transformation operation over this RDD. UserTopKRDD stores the K most similar users.

During each update step in CIP-U, after Q consumption events, the new events are stored into a DeltaProfiles RDD, which is broadcast to all the executors using the *broadcast* abstraction of Spark. Then, the hammock pairs between users are updated (in UserSimRDD) and consequently transformed to pairwise user similarities using Eq. 1. Finally, CIP-U updates the top-K neighbors (UserTopKRDD) based on the updated similarities.

RDDs for CIP-I Two Primary RDDs Are Used. ItemSimRDD stores *score* values between items. The pairwise item similarities are computed using a transformation operation over this RDD. ItemTopKRDD stores the K most similar items for each item based on the updated similarities.

During each update step in CIP-I, the item scores are updated incorporating the received CIP using Algorithm 1 in the ItemSimRDD, and consequently the pairwise item similarities are also revised using Eq. 2. CIP-I computes the top-K similar items and updates the ItemTopKRDD at regular intervals.

RDDs for DeepCIP. We implement the DeepCIP using the DeepDist deep learning framework [11] which accelerates model training by providing asynchronous stochastic gradient descent (Downpour-SGD) for Spark data. DeepCIP implements a standard master-workers parameter server model [10]. On the master node, the CIPsRDD stores the recent CIPs aggregated from the user transaction logs preserving the consumption order. Worker nodes fetch the model from the master before processing each partition, and send back the gradient updates. The master node performs the stochastic gradient descent asynchronously using the updates sent by the worker nodes. Finally, DeepCIP predicts the most similar items to a given user, based on its most recent CIP.

3.1 Experimental Setup

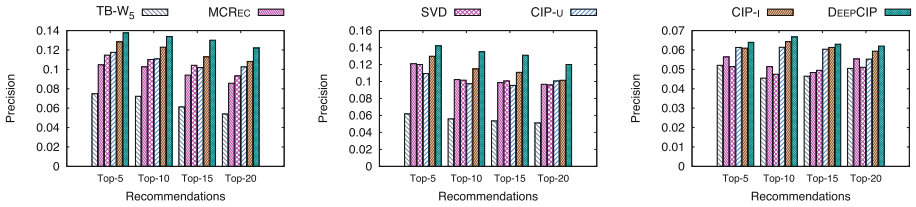
For our experiments, we use a deployment of the Spark large-scale processing framework [15]. We launch Spark as Standalone, with 19 executors each with 5 cores for a total of 96 cores in the cluster.

We then use the Grid5000 testbed to launch a Spark cluster consisting of 20 machines on Hadoop YARN, for the scalability experiments. Machines host an Intel Xeon CPU E5520@ 2.26 GHz.

Datasets and Evaluation Scheme. We use real-world traces from the Movielens movie recommendation website (ML-100K, ML-1M) [16], as well as from the Ciao [17] product review website. Those traces contain users' ratings for movies they enjoyed (ratings vary from 1 to 5). Note that the ratings are only leveraged for the explicit (rating-based) SVD recommender we use as a competitor.

The dataset is sorted based on the Unix timestamps associated with the rating events. Then, the sorted dataset is replayed to simulate the temporal behavior of users. We measure the recommendation quality as follows: we divide the sorted dataset into a *training set*, a *validation set* and a *test set*. The training set is used to train our CIP based models, whereas the validation set is used to tune the hyper-parameters of the models. For each event in the test set (or rating when applied to the explicit recommender), a set of top recommendations is selected as the *recommendation set* with size denoted as N.

Competitors. We compare the recommendation quality of our three algorithms with the following three competitors:



(a) ML-100K (b) ML-1M (c) Ciao

Fig. 1. Result quality (precision) for CIP-based algorithms and competitors.

Matrix factorization (SVD). Matrix factorization techniques map both users and items to a joint latent factor space, such that ratings are modeled as inner products in that space. We use a publicly available library (Python-recsys [18]) for evaluations.

Implicit time-based recommender (TB-W$_5$). A popular time-based recommender is providing recommendations without the need for explicit feedback [6]. Pseudo ratings are built from the collected implicit feedback based on temporal information (*user purchase-time* and *item launch-time*). We use the best performing variant: W_5 (fine-grained function with five launch-time groups and five purchase-time groups).

Markov chain-based recommender (MCRec). We compare with a recent recommender which combines matrix factorization and Markov-chains [7] to model personalized sequential behavior. We use a publicly available library [19] for the evaluation. We do not compare with FISM [3], as it is empirically shown to be outperformed by the Markov-chain based algorithm [7].

3.2 Comparison with Competitors

We refer to our technical report [20] for an in-depth exploration of parameters for our three CIP based algorithms. We obtained the following optimal setting for the hyper-parameters of those algorithms. For CIP-U: we set $\delta_H = 10$ for ML-100K, $\delta_H = 30$ for ML-1M, and $\delta_H = 10$ for Ciao to attain the best possible quality; model size is set to $K = 50$. For CIP-I we set $\delta = 1$ min for ML-100K, $\delta = 1$ min for ML-1M, and $\delta = 100$ min for Ciao; model size is set to $K = 30$. Finally for DeepCIP we set $\delta = 1$ min for ML-100K, $\delta = 1$ min for ML-1M, and $\delta = 100$ min for Ciao. We set the window size (W) to 5 for all three datasets.

The recommendation quality of all six evaluated algorithms in terms of precision ($N = 10$) is shown in Fig. 1. We draw the following observations:

(a) Regarding our three algorithms, DeepCIP always outperforms CIP-I, which in turn is always outperforming CIP-U (except on the Top-5 result on the Ciao dataset, which is due to the relatively limited number of recommendations).

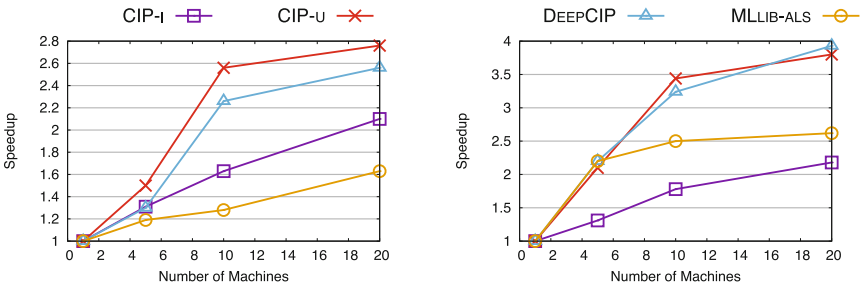
(b) The CIP based algorithms outperform TB-W_5 on all three datasets. For example, consider the top-10 recommendations in the ML-1M dataset: CIP-U provides around 1.82× improvement in the precision, CIP-I provides around 2.1× improvement, and DEEPCIP provides around 2.4× improvement.

(c) The CIP-U algorithm performs on par with MCREC, as well as with the SVD technique. CIP-I overcomes MCREC on all three scenarios, sometimes only by a short margin (ML-1M). Most notably, DEEPCIP outperforms all other approaches significantly. For example, consider the top-10 recommendations in the ML-1M dataset: DEEPCIP provides 2.4× improvement over TB-W_5, 1.29× improvement over MCREC, and 1.31× improvement over the matrix factorization algorithm. The reason behind this improvement is that DEEPCIP considers, for any given item, the *packs* of items at a distance dependent on the defined window size, whereas MCREC only considers item pairs in the sequence of chain states (*i.e.*, has a more constrained learning). Note that the precision of the SVD algorithm on Movielens (11% to 12%) is consistent with other standard quality evaluation benchmarks for state-of-the-art recommenders [21].

These results show the existence of the latent information contained in closely consumed items, accurately captured by the CIP structure. It is consistent for DEEPCIP to perform well in this setting: the original WORD2VEC concept captures relations among words w.r.t. their proximity in a given context. DEEPCIP captures item proximity w.r.t. their consumption time.

3.3 Scalability of the CIP Based Algorithms

We evaluate the scalability of our algorithms while increasing the Spark cluster size from one machine to a maximum of 20 machines. Furthermore, we also compare the speedup achieved by a matrix factorization technique (ALS)



(a) ML-100K (b) ML-1M

Fig. 2. Spark cluster size effects on computation speedup.

implemented in the publicly available MLlib library for Spark. We use 50 Spark partitions.[1]

Figure 2 depicts a sublinear increase in speedup while increasing the number of machines, on both datasets. The sublinearity in the speedup is due to communication overheads in Spark with the increasing number of machines. The speedup on ML-1M is higher due to more computations being required for larger datasets and higher utilization of the cluster. The speedup for CIP-I is similar for both datasets as its time complexity depends on the CIP size (Algorithm 1). DeepCIP scales well due to the distributed asynchronous stochastic gradient descent (Downpour-SGD) for training the Skip-gram model, where more gradient computations are executed asynchronously in parallel with the increasing number of nodes. CIP-U and DeepCIP scale better than ALS.

4 Related Work

CIP-based algorithms belong to the category of recommenders using implicit feedback from users. HOSLIM [1] proposes to compute higher order relations between items in consumed itemsets; those relations are the ones that maximize the recommendation quality, but without notions of temporality in item consumption. The proposed algorithm is time-agnostic, and does not scale for orders superior to pairs of items. Moreover, it is not designed to efficiently incorporate freshly consumed items and faces computational intractability. Barkan et al. present Item2Vec in their technical report [2], that also uses skip-gram with negative sampling to retrieve items' relations w.r.t their context in time. Besides the fact that their implementation does not scale on multiple machines due to the use of synchronous stochastic gradient descent, the technical report evaluates algorithms on private datasets. Implicit feedback has been used for multiple applications: *e.g.*, in search engines, where clicks are tracked [22]. SPrank [23] leverages semantic descriptions of items, gathered in a knowledge base available on the web. Koren et al. [24] have shown that implicit TV switching actions are valuable enough for recommendation. Within implicit based recommenders, the notion of "time" has been exploited in various ways since it is a crucial implicit information collected by all services. Baltrunas et al. presented a technique [25] similar to CIP where a user profile is partitioned into micro-profiles; still, explicit feedback is required for each of these micro-profiles. Time window (or decay) filtering is applied to attenuate recommendation scores for items with a small purchase likelihood at the moment a user might view them [26]. While such an approach uses the notion of time in transaction logs, it still builds on explicit ratings for computing the basic recommendation scores. Finally, Lee et al. [6] introduced a fully implicit feedback based approach, that weights new items if users are sensitive to the item's launch times; we compared to [6] and demonstrated a better performance.

[1] Please refer to our technical report [20] for a detailed study of the scalability of CIP based algorithms facing a varying number of partitions.

5 Conclusion

In an effort for a detailed and scalable proposal for generalizing such a direction, we presented two memory-based and one model-based recommendation algorithms exploiting the implicit notion of *consumed item packs*. We made them available on GitHub [14]. We have shown this framework to incorporate a state-of-the-art approach. In our experiments, CIP based algorithms provided a better recommendation quality than the widespread SVD-based approach [4], as well as implicit ones leveraging consumption times [6] or consumption sequences [7]. Importantly for deployments, those fits the incremental nature of collected data, to leverage freshly consumed items.

References

1. Christakopoulou, E., Karypis, G.: HOSLIM: higher-order sparse linear method for top-n recommender systems. In: PAKDD (2014)
2. Barkan, O., Koenigstein, N.: Item2vec: neural item embedding for collaborative filtering. CoRR abs/1603.04259 (2016)
3. Kabbur, S., Ning, X., Karypis, G.: FISM: factored item similarity models for top-n recommender systems. In: KDD (2013)
4. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
5. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR abs/1301.3781 (2013)
6. Lee, T.Q., Park, Y., Park, Y.-T.: An empirical study on effectiveness of temporal information as implicit ratings. Expert. Syst. Appl. **36**(2), 1315–1321 (2009)
7. McAuley, J., Ruining, H.: Fusing similarity models with Markov chains for sparse sequential recommendation. In: ICDM (2016)
8. Boutet, A., Frey, D., Guerraoui, R., Kermarrec, A.-M., Patra, R.: HyRec: leveraging browsers for scalable recommenders. In: Middleware (2014)
9. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C.: A neural probabilistic language model. J. Mach. Learn. Res. **3**, 1137–1155 (2003)
10. Dean, J., et al.: Large scale distributed deep networks. In: NIPS (2012)
11. DeepDist: lightning-fast deep learning on spark. http://deepdist.com/
12. Fontenla-Romero, Ó., Guijarro-Berdiñas, B., Martinez-Rego, D., Pérez-Sánchez, B., Peteiro-Barral, D.: Online machine learning. In: Efficiency and Scalability Methods for Computational Intellect, p. 27 (2013)
13. Chen, C., Yin, H., Yao, J., Cui, B.: TeRec: a temporal recommender system over tweet stream. In: VLDB (2013)
14. CIP-based implicit recommenders: GitHub code repo. https://github.com/rpatra/CIP
15. Apache spark. https://spark.apache.org/
16. Movielens. http://grouplens.org/datasets/movielens/
17. Ciao. http://www.ciao.com/
18. Python recsys. https://pypi.python.org/pypi/python-recsys/0.2
19. Sequence-based recommendations: GitHub code repo. https://github.com/rdevooght/sequence-based-recommendations
20. Guerraoui, R., Le Merrer, E., Patra, R., Vigouroux, J.: Sequences, items and latent links: recommendation with consumed item packs. CoRR abs/1711.06100 (2017)

21. Cremonesi, P., Koren, Y., Turrin, R.: Performance of recommender algorithms on top-n recommendation tasks. In: RecSys (2010)
22. Craswell, N., Szummer, M.: Random walks on the click graph. In: SIGIR (2007)
23. Ostuni, V.C., Di Noia, T., Di Sciascio, E., Mirizzi, R.: Top-n recommendations from implicit feedback leveraging linked open data. In: RecSys (2013)
24. Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: ICDM (2008)
25. Baltrunas, L., Amatriain, X.: Towards time-dependant recommendation based on implicit feedback. In: CARS (2009)
26. Gordea, S., Zanker, M.: Time filtering for better recommendations with small and sparse rating matrices. In: Benatallah, B., Casati, F., Georgakopoulos, D., Bartolini, C., Sadiq, W., Godart, C. (eds.) WISE 2007. LNCS, vol. 4831, pp. 171–183. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76993-4_15