From Head to Long Tail: Efficient and Flexible Recommendation Using Cosine Patterns

Yaqiong Wang[1] Junjie Wu[2] Zhiang Wu[3] Gediminas Adomavicius[1]

[1]Carlson School of Management, University of Minnesota
[2]School of Economics and Management, Beihang University
[3]Jiangsu Provincial Key Lab of E-Business, Nanjing University of Finance and Economics

Abstract

With the increasing use of recommender systems in various application domains, many algorithms have been proposed for improving the accuracy of recommendations. Among a number of other dimensions of recommender systems performance, *long-tail* (niche) recommendation performance remains an important challenge, due in large part to the *popularity bias* of many existing recommendation techniques. In this study, we propose CORE, a cosine-pattern-based technique, for effective long-tail recommendation. Comprehensive experimental results compare the proposed approach to a wide variety of classical, widely-used recommendation algorithms and demonstrate its practical benefits in accuracy, flexibility, and scalability, in addition to the superior long-tail recommendation performance.

Keywords: Recommender Systems, Pattern-Based Recommendations, Cosine Patterns, Long-Tail Recommendations

1 Introduction and Motivation

The development and adoption of web technologies and services gave rise to so-called "infinite inventory" digital commerce and content delivery platforms (Anderson 2006), such as Amazon, Netflix, and Spotify, which are able to provide many more items than their brick-and-mortar counterparts, and also fueled the development of various business analytics applications for targeting customers more effectively and for providing better services to them. Recommender systems play a unique role in online business, as high-quality personalized recommendations have been shown to have huge impact on users' purchase and consumption decisions (Pathak et al. 2010). For example, 60% of Netflix rentals and 35% of Amazon's sales are attributed to their recommendation systems (Hosanagar et al. 2013); also, more than 40% of users on Spotify continuously listen to personalized playlists generated by the platform (Buskirk 2016). Over the years, a wide variety of methods, typically based on collaborative filtering or content matching techniques, have been proposed to improve the relevance of recommended items (Adomavicius and Tuzhilin 2005, Ricci et al. 2015), focusing largely on predictive-accuracy-oriented performance metrics. However, there is a growing understanding that the quality of recommendations can be evaluated along a number of dimensions, and relying on the accuracy of recommendations alone can be insufficient for providing the most advantageous items for each user. For instance, recommending items that are already well-known and bestselling (and that the users are likely to be aware of) arguably might be less valuable than the ability to find something truly relevant and personalized from the "long tail" of the item popularity distribution. The latter constitutes the focus of our study.

Although some studies have argued that the so-called 80/20 rule, i.e., that more than 80% of the demand will be attracted by the top 20% of the products, will weaken in online markets (Anderson 2006), other studies find the opposite. For example, Tan and Netessine (2009) show that the share of demand for the top 20% of movies has increased over time from 86% to 90%. Recommender systems are known to exacerbate this problem. Due to the typically skewed item consumption distribution, it has been shown that popular items tend to be recommended disproportionately more to users (Fleder and Hosanagar 2007, Zhang 2009), partly because it is harder to predict user preferences for more idiosyncratic, less popular items. Thus, recommender systems would create rich-get-richer effects for popular products and vice versa for unpopular ones (Fleder and Hosanagar 2009, Jannach et al. 2013), and accurately recommending the long tail (niche) items remains an important challenge.

The *popularity bias* of traditional recommender systems focuses users' demand mainly to top percentile products or services, and the profit from popular items is definitely important for online businesses. However, sales of niche items could also grow to take up a large share of total sales (Anderson 2006, Hart 2007), and the long-tail (niche) titles can increase both consumer and producer surplus (Brynjolfsson et al. 2006). For example, consumers are known to have a propensity to seek variety over time (Farquhar and Rao 1976, Pessemier 1978), while online stores could benefit from recommending niche items by encouraging users to try products that are outside their awareness (Brynjolfsson et al. 2011) and thus better satisfy customers' heterogeneous needs, which is known to increase overall demand (Baumol and Ide 1956, Kekre and Srinivasan 1990). For platforms like Netflix, recommending users to rent niche movies could also lower the cost of licensing blockbusters (Goldstein and Goldstein 2006).

To alleviate the popularity bias caused by accuracy-oriented evaluation of recommendations, other performance aspects of recommender systems, e.g., diversity and novelty, have been studied in prior work (Castells et al. 2015). Novelty and diversity of recommendations could be enhanced by re-ranking the initial recommendation list (Ziegler et al. 2005, Zhang 2009, Levy and Bosteels 2010, Adomavicius and Kwon 2012) or by optimizing the ranking process using a combined objective of accuracy and diversity (Yin et al. 2012, Shi 2013, Hurley 2013, Su et al. 2013). While improvements in recommendation diversity and novelty may be associated with better long-tail recommendation as well, it's not guaranteed to be so. For example, some studies find that recommender systems can improve individual diversity while still reducing aggregate diversity (Fleder and Hosanagar 2009). Long tail recommendation could also be achieved by improving rating estimation specifically for niche items (Park and Tuzhilin 2008, Zhang and Hurley 2009, Niemann and Wolpers 2013). However, such methods tend to achieve this at the substantial expense of overall recommendation accuracy or by requiring a much richer set of features and extra preprocessing. Meanwhile, as the user population and item catalog in the system grow over time, scalability of long-tail recommendation is also an important concern. Also, the ability to easily adjust (parameterize) the degree of popularity of recommended items is another highly practical and important characteristic that is not present in a number of long-tail recommendation approaches.

Pattern-based, especially association rule-based, recommendation algorithms have attracted some attention since the early days of recommender systems research (Mobasher et al. 2001, Lin et al. 2002).

One key reason is the interpretability of their recommendations ("people who bought these items also bought..."). Typically, pattern-based algorithms first build a knowledge base containing patterns (e.g., association rules or itemsets) of typically co-occurring items and then recommend items to users based on this knowledge. Different target items are ranked by some interestingness measures, such as *support* or *confidence*, of the discovered patterns (Zaíane 2002, Kazienko 2009, Ghoshal and Sarkar 2014). Many websites have embedded rule-based approach into their commercial recommender systems, e.g., YouTube used association rules to recommend relevant videos to their users (Davidson et al. 2010). However, the traditional framework for association rule discovery, which is based on confidence and support metrics, has certain limitations that can lead to less accurate recommendations (especially with skewed data distributions). First, confidence as an interestingness measure might fail to filter some spurious associations among items (Brin et al. 1997), and recommendations based on spurious rules would not be useful. Second, support-based (i.e., frequency-based) pruning strategy of pattern mining might be problematic on data with inherently skewed support distributions, i.e., where correlations among niche items are key to long-tail recommendation yet are harder to be discovered due to low rates of niche item occurrence. As a result, traditional pattern-based recommender systems built on the support-confidence framework might not be perfectly suitable for long-tail recommendation.

To address some of these limitations, in this study we propose the CORE (COsine pattern-based REcommendation) approach to long-tail recommendation. CORE is a pattern-based method, which finds associations, represented by *cosine patterns*, among different items (especially niche items) and then utilizes the discovered associations for the purpose of item recommendation. The scalability of CORE is facilitated by a specialized data structure as well as parallel computing schemes, which is crucially important for real-time recommendation capabilities in large-scale applications.

Specifically, this paper makes the following contributions. First, we overview several pattern-based recommendation methods and discuss their limitations. In contrast, the proposed approach uses an extra measure, i.e., *cosine*, to select more relevant patterns for recommendation. Second, we develop CORE, a new recommendation method based on *cosine patterns*, which is able to substantially improve both long-tail and accuracy performance of pattern-based approaches by limiting the discovery of spurious patterns. The proposed method also supports convenient parameterization of the popularity of recommended items; that is, the dual configuration of *support* and *cosine* thresh-

olds adds more flexibility in generating recommendations of different popularity (or long tail) levels in order to achieve various recommendation goals. Third, to facilitate the scalability of the proposed approach, we further use a special CP-tree (*Cosine-Pattern tree*) structure to accelerate the recommendation process. The CP-tree is easily built and stored, and can be partitioned to parallelize the recommendation process. Based on the proposed data structure, we also design a parallel computing and load balancing framework for recommendation generation to guarantee the scalability of CORE. Fourth, comprehensive experiments on three benchmark data sets illustrate the advantages of CORE to existing pattern-based methods in terms of accuracy and long-tail recommendation as well as the advantages of CORE with respect to a number of classical, widely used collaborative-filtering-based approaches. And, a separate experiment on a larger-scale dataset further emphasizes the applicability and scalability of the proposed approach for practical, real-time recommendation tasks.

2 Background and Related Work

2.1 Recommender Systems and the Long-Tail Challenge

With the boom of the consumer-oriented content delivery and retail platforms since early 2000s, recommender systems have been progressively developed for various application domains, including movies, music, books, etc., to confront information overload and facilitate personalized information retrieval (Ricci et al. 2015). Over the years, *accuracy* of recommender systems has been the major lens through which their performance is evaluated and compared. For example, Netflix held an open competition (with $1M prize for the winner) for the most accurate recommendation algorithm to predict user ratings for movies (Koren et al. 2009). However, research studies increasingly point out that focusing on accuracy alone in recommender systems can result in sales diversity reduction (Fleder and Hosanagar 2007), since classical collaborative-filtering-based methods tend to disproportionately recommend popular items. According to Hart (2007), taking advantage of the long-tail market is one of the keys towards increasing profits on e-commerce platforms. Thus, in this study, we focus on the *long-tail* perspective of recommender systems with the goal of developing a recommendation method that can achieve better long-tail performance while still being highly competitive in terms of recommendation accuracy.

Previous studies have attempted to address the long-tail challenge in different ways. One general stream of research has focused on taking a broader perspective on recommender systems evaluation,

rather than focusing just on accuracy, which gave rise to a number of additional recommendation performance dimensions, such as diversity, novelty, etc. (Castells et al. 2015). In particular, different metrics related to recommendation diversity and novelty have been proposed, e.g., average individual (intra-list) diversity (Zhang and Hurley 2008, Ziegler et al. 2005), aggregate diversity (Fleder and Hosanagar 2009), serendipity (Murakami et al. 2007, Zhou et al. 2010), unexpectedness (Adamopoulos and Tuzhilin 2015), as well as recommendation algorithms for improving these metrics (Zhang 2009, Adomavicius and Kwon 2012, Adamopoulos and Tuzhilin 2015). Of course, diversity and novelty metrics represent an indirect way to affect long-tail recommendation performance; i.e., although these metrics have some correlation with long-tail recommendation performance (as they typically affect the distribution of recommended items), it is not guaranteed to be the case. For example, some studies show that aggregate diversity of recommendations can decrease even when individual diversity increases (Fleder and Hosanagar 2009, Jannach et al. 2013).

Therefore, some other studies have investigated ways to tackle the long-tail recommendation problem more directly. These studies can be further categorized by whether the long-tail-aware computations are applied as a pre-processing step vs. embedded more directly into the recommendation process. As an example of a pre-processing approach, Park and Tuzhilin (2008) proposed to first split items into head- and tail-groups based on their rating frequency, cluster tail items into different clusters, and then predict user ratings within each cluster. In their followup study (Park 2013), different head-tail grouping strategies to enhance long-tail recommendation are compared, and the results show that accuracy of the long-tail item recommendations indeed increases through clustering. A similar idea was explored by Zhang and Hurley (2009), where items in each active user's profile are clustered first, and recommendations are generated based on each cluster instead of complete user profiles. In contrast to pre-processing approaches, several other studies propose to embed information about item popularity into recommendation generation process, e.g., discounting popular items when learning to rank. For instance, in order to promote long-tail items in recommendations, probability for a user to consume a certain item (i.e., recommendation score) based on the whole user-item interaction graph could be discounted by the rating frequency of that item (Yin et al. 2012). Similarly, Shi (2013) proposes a Markovian graph-based recommendation approach, where weights on edges could be tuned to enhance the probability of recommending long-tail items. Other related studies propose to optimize the recommendation list based on different objectives, e.g., increasing

accuracy, reducing popularity (Hamedani and Kaedi 2019), or prioritizing niche items based on their usage context (e.g., co-occurrence with other popular items) (Niemann and Wolpers 2013).

Another set of studies propose *hybrid* approaches to improve long-tail recommendation performance. For example, in Alshammari et al. (2017), content-based and collaborative-filtering recommendation methods are used in combination to recommend long-tail and popular items, respectively. Similarly, in Zhang et al. (2012) and Ribeiro et al. (2015), the ensemble of outputs of multiple recommendation algorithms are used to balance accuracy and novelty. Other related studies use side information (i.e., information other than user-item interactions) to direct long-tail recommendation. Examples include adding semantic knowledge extracted from content information to better represent long-tail items (Craw et al. 2015) or explicitly collecting users' preferences for different types of items (Taramigkou et al. 2013).

In this paper, we focus on a new pattern-based recommendation method that tries to avoid some of the limitations of existing long-tail recommendation approaches (such as requiring a much richer set of features, significant pre-processing, or resulting in substantial reductions in accuracy), while exhibiting scalability, flexibility, and explainability benefits.

2.2 Association Analysis and Its Application to Recommender Systems

Association mining or frequent pattern discovery (Agrawal et al. 1993, Agrawal and Srikant 1994, Ceglar and Roddick 2006) is known as the task of discovering co-occurrences between items, which has its roots in the analysis of shopping basket data to better understand consumer purchasing behavior. More formally, given a database of all users' consumption histories $\mathcal{T}$, $C_u \in \mathcal{T}$ is a set of items $\{i_1, i_2, \cdots, i_{|C_u|}\}$ consumed by the user u. A pattern can be represented either by an itemset or a rule. *Itemset* P is simply a set of items, i.e., $P = \{i_1, i_2, \cdots, i_K\}$ that represents some co-occurence relationship among the items. *Rule* $P \rightarrow Q$ is an association between two disjoint itemsets that can be interpreted as an if-then statement, i.e., if P happens, then Q happens as well.

Many different measures have been proposed to evaluate the pattern strength or interestingness (Tan et al. 2002), and *support* is one of the most fundamental, popular measures. The support of itemset P is given by $supp(P) = \sigma(P)/|\mathcal{T}|$, where $\sigma(P)$ is the support count of P defined as the number of users u for whom $P \subseteq C_u$ (i.e., users who consumed all items in P). For example, $supp(P) = 0.3$ means 30% of users have consumed all items that are present in pattern P. Similarly, the support of rule $P \rightarrow Q$ is defined as $supp(P \rightarrow Q) = \sigma(P \cup Q)/|\mathcal{T}|$. In summary, the support

measure reflects the prevalence of a pattern in the data. In addition to the use of support as the pattern prevalence metric, rule discovery commonly uses an additional metric, i.e., *confidence*, which is calculated as $conf(P \rightarrow Q) = supp(P \rightarrow Q)/supp(P)$, higher confidence indicating stronger association between P and Q. For example, $conf(P \rightarrow Q) = 0.6$ means that, out of all users who consumed all items in P, 60% of them also consumed all items in Q. Confidence could also be interpreted as conditional probability, i.e., users who consume items in P tend to also consume items in Q with a probability of 60%.

If the support of itemset P satisfies a prespecified minimum support threshold τ_s, i.e., $supp(P) \geq \tau_s$, then P is a *frequent* itemset or pattern. Similarly, if the support and confidence of the rule $P \rightarrow Q$ satisfy pre-specified minimum support and confidence thresholds τ_s and τ_{conf}, i.e., $supp(P \rightarrow Q) \geq \tau_s$ and $conf(P \rightarrow Q) \geq \tau_{conf}$, then $P \rightarrow Q$ will be a discovered association rule. These thresholds can be set by users or domain experts, and several algorithms, e.g., *Apriori* (Agrawal et al. 1993) and *FP-growth* (Han et al. 2000) have been proposed to discover frequent patterns and association rules efficiently from data.

Association-based approaches have been used in many application domains, including recommender systems, for their high interpretability (Sarwar et al. 2000, Lin et al. 2002, Davidson et al. 2010). For example, in an e-commerce application, the discovered rule *"computer, monitor → keyboard, mouse"* could be used to recommend a keyboard and a mouse to consumers who already have a computer and a monitor in their shopping carts.

However, the traditional association rule discovery framework based on support and confidence has certain limitations that make it less appealing for recommendation and, more specifically, long tail recommendation. In particular, the *confidence* metric often might not reflect a meaningful association among items, in part due to item-popularity-related issues, as illustrated by the classic "coffee-tea" example (Brin et al. 1997, Tan et al. 2006). Consider a scenario where 90% of shopping baskets have coffee, i.e., in general consumers buy coffee 90% of the time. Furthermore, let's assume that, among all baskets containing tea, 75% of them contain coffee as well, i.e., the confidence of rule *tea → coffee* is 75%. In other words, even though the confidence of buying coffee given that tea is already in the shopping basket is quite high (i.e., 75%), the two items are actually negatively associated with each other, i.e., buying tea reduces users' probability of buying coffee. Thus, recommending based on such rules is likely to reduce recommendation accuracy.

Table 1: Impact of null-consumptions on pattern/rule evaluation measures.

Data Set	A	B	AB	$\overline{A}B$	$A\overline{B}$	$\overline{AB}$	$conf(A \rightarrow B)$	$lift(A \rightarrow B)$	$cos(\{A, B\})$
D_1	11,000	15,000	10,000	5,000	1,000	100,000	0.91	7.03	0.78
D_2	11,000	15,000	10,000	5,000	1,000	100	0.91	0.98	0.78

A: # of users who consumed A; AB: # of users who consumed A and B; $\overline{AB}$: # of users who consumed neither A nor B; B: # of users who consumed B; $\overline{A}B$: # of users who consumed B not A; $A\overline{B}$: # of users who consumed A not B.

To deal with the aforementioned limitation, correlation-oriented measures can be used to augment the support-confidence framework for association rules (Tan et al. 2006). Among such measures, *lift* has been a popular choice. Lift of rule $P \rightarrow Q$ is calculated as $lift(P \rightarrow Q) = conf(P \rightarrow Q)/supp(Q) = \frac{supp(P \rightarrow Q)}{supp(P) * supp(Q)}$, which reflects the degree to which the occurrence of P "lifts" the occurrence of Q. However, the lift metric is sensitive to *null-consumption* histories, i.e., to the number of users who did not consume *any* items contained in the rule of interest. As a quick illustration, Table 1 shows two example data sets about consumption of items A and B, including statistics on number of users who consumed both A and B (denoted as AB), not A but B ($\overline{A}B$), A but not B ($A\overline{B}$), neither A nor B ($\overline{AB}$). The latter value represents the number of null-consumption histories with respect to A and B. The table shows that $lift(A \rightarrow B)$ changes significantly with the size of $\overline{AB}$, even when the consumption statistics of A, B, and AB are identical. In summary, for any general pattern $P \rightarrow Q$, such sensitivity of a key association metric (i.e., lift) to null-consumption histories might not be desirable, as the metric becomes more reflective of the prevalence of the pattern in data (which is already captured by support) than of the association between P and Q.

Another limitation of the support-confidence framework is that the traditional support-based pruning strategy for pattern mining (based on setting an appropriate support threshold) might be inadequate on data with skewed support distributions (Xiong et al. 2006). Lower thresholds often result in excessive or redundant patterns, as well as patterns that have items with substantially different support levels (i.e, the so-called cross-support patterns, to be discussed in Section 2.3), leading to lower-quality recommendations. Higher thresholds favor highly frequent items and omit less frequent but potentially advantageous itemsets (patterns), which leads to popularity bias.

To address the limitations of confidence and lift metrics, we propose to use *cosine* patterns for recommendation. In what follows, we give some preliminaries on cosine patterns and describe important properties that make them particularly suitable for long tail and scalable recommendation.

2.3 Cosine Patterns and Their Properties

In this paper, we adopt *cosine* (Tan et al. 2002) as an interestingness measure to be used simultaneously with support for pattern evaluation and pattern-based recommendation. The *cosine* value of a K-itemset (i.e., an itemset of K items) P is defined as:

$$cos(P) = \frac{supp(P)}{\left(\prod_{k=1}^{K} supp(\{i_k\})\right)^{1/K}},\ K \geq 2, \tag{1}$$

which reduces to the traditional *cosine* measure when $K = 2$ (Wu et al. 2012). An important advantage of the cosine metric, as can be seen from the example in Table 1, is that the cosine value of a pattern is not affected by the number of null-consumptions.

In traditional association analysis, itemset P is called a *frequent* pattern if $supp(P) \geq \tau_s$, where τ_s is the user-defined minimum support threshold. Cosine pattern discovery takes one more threshold τ_c, i.e., the minimum cosine threshold, for pattern evaluation.

Definition 2.1 (Cosine Pattern)**.** Itemset P is a cosine pattern w.r.t. τ_s and τ_c, if $supp(P) \geq \tau_s$ and $cos(P) \geq \tau_c$, where τ_s and τ_c are user-defined thresholds.

Intuitively, a higher cosine value reflects a more "cohesive" pattern (i.e., a stronger association between items in the pattern). More specifically, the key appeal of the cosine measure lies in its *anti-cross-support* property. As defined by Xiong et al. (2006), P is a *cross-support pattern* (CSP) w.r.t. τ ($0 \leq \tau \leq 1$) if its CSP value $V_{csp}(P) \leq \tau$. Here $V_{csp}(P) = s(i_l)/s(i_h)$, with i_l and i_h representing items with lowest and highest support values in P, respectively. A smaller $V_{csp}(P)$ value indicates a more severe imbalance of item supports in a pattern. By its definition, a CSP is a pattern containing items with significantly different support levels and, thus, more likely representing spurious, less cohesive (and potentially less meaningful) associations among items, as will be discussed in more detail in Section 3.1. The minimum *confidence* threshold, which is traditionally used in association rule mining, is not sufficient for filtering out CSPs. It could be argued that CSPs can be filtered by setting a higher τ_s, but this would lead to the omission of rare but interesting (niche) patterns and, thus, to the information loss for recommendation, especially for long tail recommendation. As shown by Wu et al. (2012), for $P = \{i_1, i_2, \cdots, i_K\}$, $cos(P) \leq \sqrt[K]{V_{csp}(P)}$. This implies that a pattern tends to have a lower cosine value as $V_{csp}(P)$ gets smaller; i.e., the patterns with lower cross-support values are less likely to be cosine patterns. Thus, cosine measure has the *anti-cross-support* property.

The anti-cross-support property of the cosine metric allows to control the "cohesiveness" of patterns discovered, making cosine patterns suitable for flexible recommendations. For example, setting large τ_s and moderate τ_c allows to obtain cohesive patterns with highly popular items; in contrast, reducing τ_s while keeping τ_c high allows to obtain the most cohesive patterns with both popular and niche items. Both cases, based on threshold use, can guard against excessive generation of redundant patterns while providing flexibility to recommend items of desired popularity levels.

Mining cosine patterns based on τ_s and τ_c thresholds is not a trivial task. As is well known, the support measure holds the so-called *anti-monotone property* (AMP) defined as follows.

Definition 2.2 (AMP)**.** Itemset interestingness measure M is said to possess **AMP**, if for every P and P' such that $P \subset P'$, we have $M(P) \geq M(P')$.

Since support holds AMP, if itemset P is infrequent (i.e., $supp(P) < \tau_s$), then *all* its supersets are also infrequent (i.e., $\forall X$ such that $X \supset P$, we have $supp(X) \leq supp(P) < \tau_s$) and can be removed from consideration without further calculations. AMP of an interestingness measure is critical for efficient pattern mining (Agrawal and Srikant 1994). While the cosine measure does not hold AMP, it holds the following *Conditional Anti-Monotone Property* (CAMP) (Wu et al. 2012).

Definition 2.3 (CAMP)**.** Itemset interestingness measure M is said to possess **CAMP**, if for every P and P' such that (i) $P \subset P'$ and (ii) $\forall\ i \in P$, $\forall\ i' \in P' \setminus P$, $s(\{i\}) \leq s(\{i'\})$, we have $M(P) \geq M(P')$.

Compared with AMP, CAMP provides an extra condition to gain the anti-monotone property. Because cosine holds CAMP, cosine patterns can be mined efficiently. In particular, if P is not a cosine pattern, then any pattern P' that can be created by adding high-support items to P (i.e., items that have higher support than any item in P) will not be a cosine pattern due to CAMP, and thus can be removed from consideration without further calculations. This principle allows to avoid extra computations and lays the foundation of efficient cosine pattern mining.

The *CoPaMi* algorithm was proposed specifically to mine cosine patterns (Wu et al. 2014). It aligns the items in each consumption history in a support-ascending order (to facilitate CAMP conditions for patterns sharing the same prefix), and then employs the tree-based data structure and depth-first traversal strategy to effectively prune patterns based on both support and cosine. We parallelize *CoPaMi* on the Spark platform and use it to mine cosine patterns in our experiments.

3 Cosine Pattern-Based Recommendation

3.1 Cosine Patterns for Recommendation

As mentioned earlier, pattern-based recommender systems have attracted substantial attention (Lin et al. 2002, Zaíane 2002, Kazienko 2009), partly for their high interpretability of recommendations. A recent survey (Paraschakis et al. 2015) on more than 30 popular e-commerce platforms also reveals that industries often favor less complex recommendation techniques like association rules mining or nearest neighbor-based collaborative filtering for efficiency and engineering cost concerns. This indicates that improving the recommendation performance of pattern-based (such as itemset- or rule-based) methods is of theoretical and practical importance.

We argue that the prevalence of the cross-support patterns is largely responsible for the lower accuracy and higher popularity bias of traditional rule-based recommender systems. To illustrate this, Table 2 shows some representative examples of 2-item movie association rules (10 out of top 150 highest confidence rules) and 2-item cosine patterns (10 out of top 150 highest cosine patterns)[1] discovered from the MovieLens data[2] with $\tau_s = 2\%$, $\tau_{conf} = 76\%$, and $\tau_c = 0.5$.[3]

From Table 2 we can see that there is a large imbalance of support levels between the antecedent and consequent in each example association rule, as indicated by low V_{csp} values. In contrast, the V_{csp} values for cosine patterns are substantially higher. Moreover, in the last two columns of Table 2, we present two additional indicators of how related are the two movies that appear in each pattern (or how "cohesive" the pattern is). The first indicator is the correlation coefficient (CorrCoef) of the ratings for two movies, i.e., how similar are the preferences for these two movies among the users who saw both of them. The second indicator is the Jaccard similarity (JSim) of the movie consumptions, i.e., how similar are the sets of users who saw each movie. Both of these indicators consistently show that association rules contain items that are less related to each other (i.e., have substantially lower CorrCoef and JSim) than cosine patterns. This insight is further emphasized by Table 3, which provides the aggregate statistics across top-150 association rules and top-150 cosine patterns. In particular, the patterns mined based on the confidence measure (i.e., association rules)

[1] For comparison, the illustrative sets of 10 cosine patterns and 10 association rules were picked to have similar *support* (and to be representative of the broad range of support values).

[2] Detailed information of this data set could be found in Table 4.

[3] Thresholds were set to have similar number (between 150 and 200) of patterns discovered in both cases.

contain items that are highly imbalanced in terms of their support and substantially less related to each other than the patterns mined based on the cosine measure.

In summary, recommendations generated from association rules have several limitations. In particular, as shown above, association rules tend to contain items that are less related to each other, which can lead to lower accuracy when deployed for recommendations. As importantly, due to the fact that many discovered association rules have low cross-support values, the movies that end up being recommended using association rules (i.e., movies in the consequent of the rule) are largely high-support items (i.e., popular movies), such as *Star Wars*, *Fargo*, *Back to the Future*, and *Raiders of the Lost Ark*. This perpetuates the so-called popularity bias existing in many collaborative-filtering-based recommender systems, leading to insufficient recommendation of niche movies and long-tail recommendation challenges. As shown in Tables 2 and 3, cosine pattern mining can effectively overcome these limitations due to its anti-cross-support objective, which allows for discovery of more cohesive patterns, including patterns with less common (niche) items. Take pattern {*Manon of the Spring, Jean de Florette*} as an example. Although movies in this pattern have been watched only by a relatively small number of users, the cohesiveness of this pattern is quite high: the movies are liked very similarly by users who saw both of them (CorrCoef = 0.76), and the sets of users who saw each movie are quite similar as well (JSim = 0.60). Nevertheless, such a pattern is unlikely to be discovered as an association rule due to its relatively low confidence. The ability to discover connections among items with comparatively smaller audience is a key to addressing the long-tail recommendation challenge and, thus, is one of the motivating factors for the proposed cosine pattern-based method for long-tail recommendation. At the same time, the minimum support and cosine threshold parameters provide the flexibility to fine-tune the proposed method to the desired specifications (e.g., in terms of recommending popular vs. niche items), as will be shown later in the paper.

3.2 Basic Recommendation Scheme

Here we introduce the basic scheme of cosine pattern-based recommendation. We assume that the set of all applicable cosine patterns $\mathcal{CP}$ (i.e., patterns with $supp(P) \geq \tau_s$ and $\cos(P) \geq \tau_c$ for some user-specified thresholds τ_s and τ_c) has been mined in advance (e.g., using the standard library *CoPaMi*) and focus on the problem of generating top-K recommendations for each user.

Given the set of all discovered cosine patterns $\mathcal{CP}$ and user u (represented by her consumption history C_u), the recommendation process consists of three main stages: (i) identifying u's target

Table 2: Examples of association rules and cosine patterns discovered from the MovieLens dataset.

Association Rules							
Antecedent	**Consequent**	**Supp (%)**	**Conf (%)**	**Cosine**	**V_{csp}**	**Corr Coef**	**JSim**
Dead Man (34)	Star Wars (583)	2.3	77.3	0.15	0.06	-0.22	0.05
Only You (39)	The Princess Bride (324)	2.9	77.8	0.24	0.12	0.06	0.10
Giant (51)	Casablanca (243)	3.9	81.1	0.33	0.21	0.21	0.20
Trees Lounge (50)	Fargo (508)	4.1	79.5	0.24	0.10	0.23	0.10
Weekend at Bernie's (60)	Back to the Future (350)	4.6	79.1	0.30	0.31	0.31	0.16
Dumb and Dumber (50)	Back to the Future (350)	4.8	82.2	0.29	0.20	-0.06	0.18
Flirting With Disaster (42)	The Empire Strikes Back (367)	5.5	77.4	0.25	0.10	0.12	0.24
Nell (81)	Raiders of the Lost Ark (420)	5.8	76.4	0.30	0.20	0.32	0.17
Victor/Victoria (77)	Return of the Jedi (507)	5.8	80.0	0.28	0.15	0.09	0.13
Casino (91)	Raiders of the Lost Ark (420)	7.0	78.8	0.34	0.22	0.14	0.20
Cosine Patterns							
Movie 1	**Movie 2**	**Supp (%)**	**Conf (%)**	**Cosine**	**V_{csp}**	**Corr Coef**	**JSim**
Manon of the Spring (58)	Jean de Florette (64)	2.3	37.9	0.51	0.38	0.76	0.60
Three Colors: White (59)	Three Colors: Blue (64)	2.9	45.8	0.87	0.42	0.75	0.58
A Grand Day Out (66)	The Wrong Trousers (118)	3.9	56.1	0.58	0.31	0.66	0.50
Private Benjamin (66)	Home Alone (137)	4.1	58.2	0.52	0.28	0.32	0.31
Bram Stoker's Dracula (120)	Interview with the Vampire (137)	4.8	38.3	0.52	0.34	0.45	0.55
Batman Forever (114)	Batman Returns (142)	4.8	35.4	0.51	0.40	0.35	0.55
Star Trek: Final Frontier (63)	Star Trek: Motion Picture (117)	5.6	45.3	0.55	0.33	0.44	0.52
Die Hard With a Vengeance (151)	Die Hard 2 (166)	5.7	35.8	0.51	0.33	0.75	0.68
Under Siege (124)	Clear and Present Danger (179)	5.8	44.4	0.51	0.31	0.50	0.47
Ghost (170)	Mrs. Doubtfire (192)	7.0	39.4	0.51	0.35	0.45	0.48

In parentheses after each movie title: the number of users who rated the movie;
V_{csp}: support ratio of items within rules/patterns;
CorrCoef: correlation coefficient of movie ratings;
JSim: Jaccard similarity of movie consumptions.

Table 3: Descriptive statistics of top-150 rules/patterns.

	Supp (%)	**Conf (%)**	**Cosine**	**V_{csp}**	**CorrCoef**	**JSim**
Association Rules	3.0 (1.08)	80.37 (3.49)	0.25 (0.05)	0.13 (0.05)	0.17 (0.22)	0.11 (0.05)
Cosine Patterns	11.6 (3.70)	42.50 (4.00)	0.53 (0.02)	0.83 (0.14)	0.32 (0.15)	0.55 (0.06)

Standard Deviation in Parentheses.

items T_u; (ii) identifying the set of eligible patterns $\mathcal{EP}_{ui}$ (where $\mathcal{EP}_{ui} \subseteq \mathcal{CP}$) for each target item $i \in T_u$; and (iii) calculating recommendation scores for each $i \in T_u$ (by aggregating information from $\mathcal{EP}_{ui}$) and ranking all target items according to the scores. We describe each stage below.

Stage 1: For each user u, target item set T_u consists of items not yet consumed by user u, i.e., $T_u = I \setminus C_u$, where I represents the set of all possible items.

Stage 2: For each target item i in T_u, we need to find eligible pattern set $\mathcal{EP}_{ui}$. An *eligible pattern* is defined as follows.

Definition 3.1 (ui-Related Eligible Pattern)**.** Let P be a cosine pattern and C_u be user u's consumption history. P is an **eligible pattern** for u w.r.t. i, if $i \in P$, $i \notin C_u$, and $P \setminus \{i\} \subseteq C_u$.

In other words, cosine pattern P is a ui-related eligible pattern if: (1) P contains target item i, and (2) all other items in P (i.e., other than i) have been consumed by user u. Thus, any ui-related eligible pattern represents a cohesive itemset consisting of some items already consumed by u and

one new item i, making item i a natural candidate for recommendation. The set of all ui-related eligible patterns is denoted as $\mathcal{EP}_{ui}$.

As a simple illustration, let's assume that we have a recommendation application with seven items, i.e., $I = \{A, B, C, D, E, F, G\}$, where the following three cosine patterns (itemsets) have been mined from users' consumption histories: $\mathcal{CP} = \{\{A, B, C\}, \{A, D, E, G\}, \{A, B, D\}\}$. Also, let's assume that user u's consumption history is $C_u = \{B, C, D, E\}$. Then, user u's target item set is $T_u = I \setminus C_u = \{A, F, G\}$. Considering item A as a target item, we can see that $\mathcal{EP}_{uA} = \{\{A, B, C\}, \{A, B, D\}\}$. In other words, $\{A, B, C\}$ is a uA-related eligible pattern, since all items in it except A have been consumed by u, and so is pattern $\{A, B, D\}$. Note, however, that $\{A, D, E, G\}$ is not an uA-related eligible pattern even though it also contains target item A, because there are multiple (i.e., more than one) target items in it (i.e., A and G).

Stage 3: For user u, recommendation score for target item i is derived from $\mathcal{EP}_{ui}$ by summing the cosine values of all patterns in $\mathcal{EP}_{ui}$, i.e., $score(u, i) = \sum_{P \in \mathcal{EP}_{ui}} cos(P)$. For any given user u, all target items in T_u will be ranked by their recommendation scores, and the recommendation list L_u for user u would be generated by selecting the top-K ranked items.

In other words, the current version of CORE adopts a relatively simple scoring method for target items $i \in T_u$, which adds up the cosine values of all ui-related eligible patterns. There have been several studies investigating ways in which pattern-based recommendation algorithms could combine eligible patterns to provide better recommendations. For instance, Wickramaratna et al. (2009) proposed a Dempster-Shaffer-based approach to combine rules with conflicting predictions. Ghoshal and Sarkar (2014) proposed to partition eligible rules into groups of rules with disjoint antecedents and same consequent and developed a probability model to select the group that maximizes the likelihood of purchasing target item for recommendation. Lin et al. (2002) adopted heuristics like adding up the supports and confidences of all eligible rules with the same consequent as its recommendation score. Even though there exist different strategies for determining recommendation scores for items based on discovered patterns, theoretical or empirical studies on deriving optimal strategies are rarely seen. In this study, we used the cosine value of a pattern, as it provides a meaningful quantification of the itemset cohesiveness, as discussed earlier. Furthermore, if target item i appears in *multiple* ui-related eligible patterns (i.e., provides a cohesive combination with several consumed itemsets of user u), arguably this provides an even stronger signal of item i's relevance to u; hence, we chose to use a

simple aggregation of the cosine values across *all* eligible patterns. This scoring approach is not only easy to implement and computationally scalable, but also demonstrates excellent recommendation performance (as demonstrated by our experimental evaluation). In-depth analysis of optimal recommendation score aggregation across multiple patterns represents an interesting direction for future work.

We call the above approach *COsine pattern-based REcommendation*, or CORE for short. As will be demonstrated in the evaluation section, by leveraging the anti-cross-support property of cosine patterns, CORE not only exhibits good recommendation accuracy, but is also able to successfully recommend long-tail items. Furthermore, the two threshold parameters for cosine pattern mining (i.e., τ_s and τ_c) provide CORE with flexibility in recommending items across the popularity distribution. For example, in order to to recommend more popular items we can set a high τ_s and a moderate τ_c, while to recommend more niche items we can set a small τ_s but a high τ_c.

3.3 Cosine-Pattern Tree Traversal Approach

The key computational challenge of the proposed cosine-pattern-based recommendation process is finding ui-related eligible patterns for given user u's all target items i. To deal with this, we propose to use a data structure called Cosine-Pattern Tree to boost the eligible pattern discovery process, which leads to CORE+ (an enhanced version of CORE).

Based on the definition, for given user u and target item $i \in T_u$, a simple way to compute $\mathcal{EP}_{ui}$ is to first find all cosine patterns that contain i, i.e., $\mathcal{CP}_i = \{P \in \mathcal{CP} | i \in P\}$, and then keep only those where the remaining items are covered by C_u, i.e., $\mathcal{EP}_{ui} = \{P \in \mathcal{CP}_i | P \setminus \{i\} \subseteq C_u\}$. The most time-consuming aspect of this calculation is determining whether pattern $P \in \mathcal{CP}_i$ satisfies the latter condition. A straightforward way to do this is to examine each item in $P \setminus \{i\}$ to see whether it is contained in C_u. By storing C_u in a hash table, where time complexity to look up any item is $\Theta(1)$, the overall time complexity for checking all items in one pattern is $O(|P|)$, and it would take $O(\sum_{P \in \mathcal{CP}_i} |P|)$ to go through all candidates and identify all ui-related eligible patterns.

Under this strategy, two factors can slow down the recommendation process. First, the target item set, i.e., number of non-consumed items, for each user is typically very large; thus, time required to find candidate and eligible patterns for all items can add up quickly. Second, due to CAMP, a cosine pattern typically contains many similar cosine patterns as subsets, which would entail a great deal of repeated (redundant) matching of highly similar patterns with C_u.

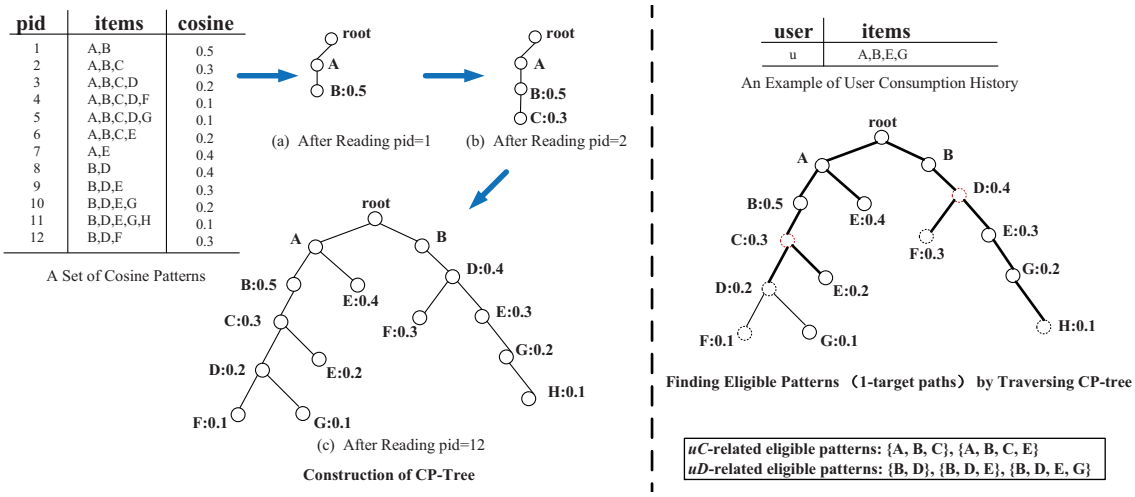


Figure 1: An example for CP-tree construction and traversal.

Both of these factors can be addressed by using advantageous data structures and algorithms for storing and retrieving cosine patterns. In particular, to facilitate efficient cosine pattern traversal and reduce redundant matching in eligible pattern detection, we build upon the notion of the FP-tree (Han et al. 2000) as well as the ideas from the *CoPaMi* cosine pattern mining approach (Wu et al. 2014) to define and use *Cosine-Pattern Tree* (CP-tree) for compact representation of cosine patterns. Based on CP-tree, an intelligent, depth-first search strategy is designed to find all eligible patterns efficiently.

Generally, CP-tree is constructed in a similar fashion as FP-tree, i.e., it is a fully-connected, hierarchical tree structure, where each node represents an item. Each path from the root node to any other node in the CP-tree represents a cosine pattern (itemset) containing all the items on that path. Thus, cosine patterns that share a common prefix will share (a part of) the same path.

More formally, CP-tree consists of two types of nodes: (i) one special node denoting the start of all paths (i.e., all patterns stored in the tree) and labeled as *root*, and (ii) possibly multiple regular nodes denoting items, each labeled with a corresponding item name. Each node has three fields, storing its relevant information: `item` (i.e., the label of this node), `cosine` (i.e., the cosine value of the pattern that ends with this node), and `childlist` (i.e., the list of children nodes that are connected to this node).

Before tree construction, all items in each cosine pattern P are sorted in a support-ascending order, which guarantees that every prefix of P is also a cosine pattern, due to CAMP. E.g., if $supp(A) \leq supp(B) \leq supp(C)$ and $\{A, B, C\}$ is a cosine pattern, then $\{A, B\}$ is guaranteed to be

a cosine pattern, due to CAMP. An illustration of CP-tree construction process is shown on the left side of Fig. 1, where all items are assumed to be sorted in the support-ascending order. Initially, the CP-tree contains only the default root node. After reading the first cosine pattern $\{A, B\}$, the nodes labeled as A and B are created, and path $root \rightarrow A \rightarrow B$ is then added, and the value of $cos(\{A, B\}) = 0.5$ is saved in the `cosine` field of node B, as shown in Fig. 1a. The second cosine pattern $\{A, B, C\}$ shares a common prefix $\{A, B\}$ with the first pattern, and therefore only one new node marked as C is added to the end of path $root \rightarrow A \rightarrow B$ with the corresponding $cos(\{A, B, C\}) = 0.3$ value in Fig. 1b. This process continues until every cosine pattern has been mapped onto one of the paths in CP-tree, which leads to the final CP-tree in Fig. 1c. Because of this specific prefix-based tree construction as well as the CAMP property of the cosine measure, all distinct cosine patterns are represented by all the paths from the root node to *every* node that is below the first two levels of CP-tree (since a cosine pattern must have at least two items).

Thus, any arbitrary path $[P] = root \rightarrow i_1 \rightarrow \cdots \rightarrow i_p$ in CP-tree represents cosine pattern $P = \{i_1, \cdots, i_p\}$, when $|P| \geq 2$. With respect to specific user u, any given path $[P]$ (and its corresponding cosine pattern P) can be classified into three different categories – 0-target path, 1-target path, and multi-target path – depending on how many target items the path contains. In particular, $[P]$ is a *0-target path* if $|P \setminus C_u| = 0$ or, equivalently, $P \subseteq C_u$, i.e., the path does not contain any target items for user u. Alternatively, $[P]$ is a *1-target path* if $|P \setminus C_u| = 1$. Finally, $[P]$ is a *multi-target path* if $|P \setminus C_u| \geq 2$, i.e., if it contains multiple target items for user u. Note that only 1-target paths represent ui-related eligible cosine patterns that can be used for recommendation, as they contain exactly one target item.

The above categorization suggests an intelligent and highly efficient computational strategy that allows, for any user u, to find all relevant target items and calculate their recommendation scores with a *single traversal through CP-tree*. Intuitively, the main idea is to traverse CP-tree, visiting nodes in a depth-first manner. Each visited node represents path $[P]$ (i.e., path from the root to this node), which can be one of the following: (i) 0-target path, in which case no recommendation decisions need to be made, and the depth-first search continues to the children of this node; (ii) 1-target path, in which case the recommendation score for target item i (that is contained in $[P]$) is increased by $cos(P)$, and the depth-first search continues to the children of this node; or (iii) multi-target path, in which case no recommendation decisions need to be made, and the depth-first search is stopped

Algorithm 1 Eligible-pattern searching and target-item scoring from CP-tree.

Input: $root$, C_u ▷ $root$ is the root node of CP-tree, C_u is the consumption history of user u
Output: $score$ ▷ list of recommendation scores of all items for user u

```
1: procedure SearchCP(root, C_u)
2:     S := ∅;                                              ▷ S: a last-in-first-out stack
3:     for i ∈ I do                                         ▷ I: the set of all items
4:         score(i) := 0;                                   ▷ initialization of recommendation score for each item
5:     target := null                                       ▷ detected target item in the beginning is null
6:     PushChildNodes(S, root, target);
7:     while S ≠ ∅ do                                       ▷ continue traversing until the stack is empty
8:         (cur, target) := S.POP;                          ▷ pop out the top tuple (current node, target item) in S
9:         if target is not null then                       ▷ check whether a target item is already detected
10:            if cur.item ∈ C_u then                       ▷ check whether current item is in user's consumption history
11:                score(target) := score(target) + cur.cosine;   ▷ update recommendation score of the target item
12:                PushChildNodes(S, cur, target);          ▷ push current node's children and current target item into the stack
13:            else
14:                continue                                 ▷ traversal of a path stops when a second target item is detected
15:        else                                             ▷ in case no target item has been detected so far
16:            if cur.item ∈ C_u then
17:                PushChildNodes(S, cur, target);          ▷ keep traversing when no target item is detected
18:            else
19:                target := cur.item;                      ▷ update the value of taget when a target item is detected
20:                if cur is not child of root then         ▷ update score only when the target item locates beyond the first level
21:                    score(target) := score(target) + cur.cosine;
22:                PushChildNodes(S, cur, target);          ▷ push current node's children and updated target item into the stack
23: end procedure
24: procedure PushChildNodes(S, cur, target)
25:     for node ∈ cur.childlist do
26:         S.Push(node, target);                            ▷ push current node's children and detected target item into the stack
27: end procedure
```

along this path, as all subsequent extensions of path $[P]$ will continue to be multi-target paths. In summary, the proposed approach does not have to check all possible target items, but rather finds them (and calculates their recommendation scores) organically in one shot by efficiently browsing cosine patterns and matching them with each user's consumption history.

Algorithm 1 provides more detailed overview of the implementation of the proposed search-and-scoring routine. The main function `SearchCP` (Lines 1-23) employs a depth-first search on CP-tree. Besides variable *cur* to indicate the current node, variable *target* is introduced to indicate the target item contained in the current eligible pattern. Traversal along any given path in CP-tree terminates as soon as the path becomes a multi-target path (Lines 13-14), which avoids unnecessary search of longer patterns. Last-in-first-out stack S is used to execute the depth-first traversal, and variable *target* corresponding to target item contained in the current path is pushed into (or popped out from) the stack simultaneously with each node (Lines 6, 8, 12, 17, 22). Lines 11 and 21 update the recommendation scores of target items whenever the traversed path is a 1-target path.

The right side of Fig. 1 illustrates the use of Alg. 1 for user u with specific consumption history C_u. Note that the cosine values of patterns are shown next to the end nodes of their corresponding paths,

the dashed nodes are target items (i.e., items that are not in C_u), and thick lines represent actual traversals performed by Alg. 1. Consider traversal along $root \rightarrow A \rightarrow B \rightarrow C$. Because C is the first item not contained in C_u, $root \rightarrow A \rightarrow B \rightarrow C$ becomes a 1-target path with C as its target item. This path can be extended further either with D or with E. In the case of former, traversal would stop at node D, as it is the second item not contained in C_u, making the path a multi-target path at that point; going deeper would not detect any new eligible patterns. This avoids redundant checks of successive nodes D, F, and G along the same path. In contrast, $root \rightarrow A \rightarrow B \rightarrow C \rightarrow E$ would be identified as another 1-target path with C as its target item. As another example, after traversal of path $root \rightarrow B \rightarrow D$, D is identified as another target item, and three 1-target paths with D as target item would be discovered: $root \rightarrow B \rightarrow D$, $root \rightarrow B \rightarrow D \rightarrow E$, and $root \rightarrow B \rightarrow D \rightarrow E \rightarrow G$. Eventually, in this example, five eligible patterns that contribute to recommending C and D are identified with a single traversal of CP-tree.

CORE+ recommendation approach is highly computationally efficient. Space complexity of CP-tree is $O(|\mathcal{CP}|)$, since the total number of nodes in the tree equals the total number of cosine patterns plus a small fixed number (i.e., the root node and its immediate children). Meanwhile, given any user consumption history C_u, time complexity of finding all eligible patterns using Alg. 1 is $O(|\mathcal{CP}|)$. To elaborate, there are $O(|\mathcal{CP}|)$ nodes in the CP-tree and, in the extreme case, all of them may have to be examined, with constant amount of time needed per node; e.g., to check whether an item is contained in C_u is $\Theta(1)$ using a hash table. Therefore, with CP-tree and Alg. 1, the time complexity of recommendation given C_u reduces substantially from $O(|T_u| \sum_{P \in \mathcal{CP}} |P|)$ to $O(|\mathcal{CP}|)$, where $|P|$ is the number of items in pattern P. This implies that the upper bound of the running time of CORE+ is simply proportional to the number of cosine patterns, regardless of their size or the number of potential target items.

3.4 Parallelizing Cosine Pattern-based Recommendation

In this section, we describe a distributed framework designed for CP-tree based recommendation, which leads to CORE++, a parallelized version of CORE+ for online recommendation.

Intuitively, *user partitioning*, i.e., distributing active users to different servers on which a complete CP-tree is stored in advance for recommendation, is a straightforward way to improve scalability. For example, according to the widely-used strategy of proxy cache servers (Gama et al. 2001), network traffic could naturally be reduced by replicating static content (CP-tree in our case). However,

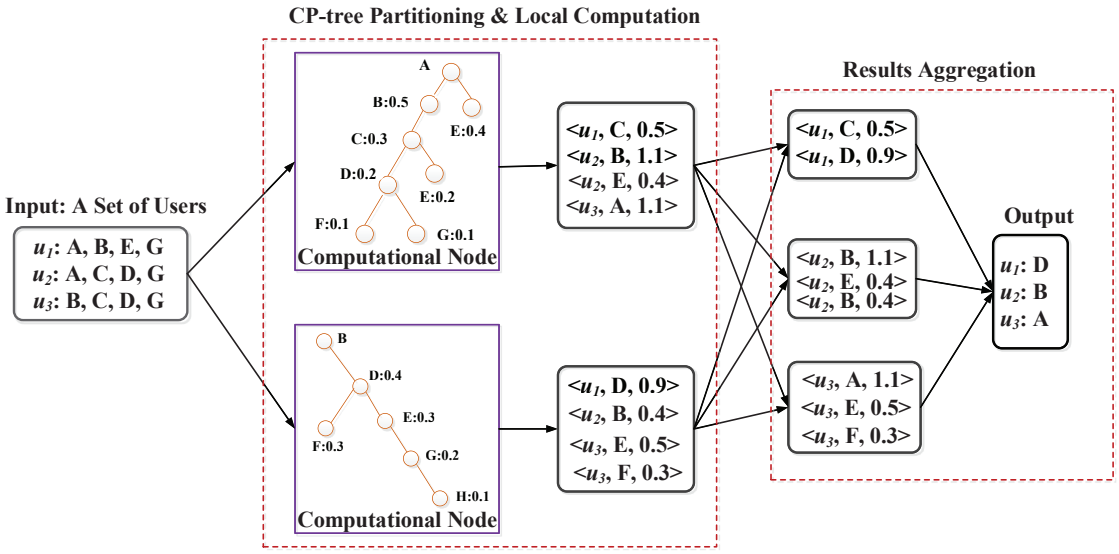


Figure 2: Illustration of parallelization based on CP-tree partitioning.

when the CP-tree is large, i.e., a huge amount of cosine patterns is discovered from consumption or purchase data of a gigantic online retailer like Amazon, recommendation generation for a single user can still take a considerable amount of time.

To address this challenge, we propose to use a parallelization framework based on *CP-tree partitioning* to further speed up recommendation. Specifically, by taking a complete CP-tree as a *forest* with a null root node, each subtree rooted at the second level can then be allocated to one computational node, and thus the entire CP-tree can be stored separately on Z available computational nodes. The parallelized CP-tree based recommendation (i.e., CORE++) could then be done in three stages: (i) broadcast user u's consumption history C_u to Z computational nodes; (ii) run Alg. 1 for u on Z nodes in parallel; (iii) aggregate local scores for each target item to obtain the final recommendation score. Fig. 2 illustrates the three stages using the example from Fig. 1, where the initial CP-tree in Fig. 1c is decomposed into two subtrees by removing the root node. Each subtree is then distributed to a separate computational node, along with all users' consumption histories. Recommendation score for each target item based on each subtree would be generated by running Alg. 1 simultaneously on the two computational nodes. The final recommendation score for each target item can then be obtained by aggregating scores from each node (e.g., recommendation score of B for u_2 based on two subtrees is aggregated as $1.1 + 0.4 = 1.5$). In Fig. 2, for each user, we pick one target item with the highest score for recommendation.

In the above parallelization scheme, the time to generate recommendations for each user hinges on the slowest computational node. Thus, it is desirable to partition the CP-tree in such a way that workload on all nodes is evenly distributed. This could be formally defined as a *load-balanced partitioning* (LBP) problem as follows.

Definition 3.2 (LBP Problem)**.** Let $\mathcal{S} = \{s_1, s_2, \cdots, s_q\}$ be the set of subtrees and $\alpha_i \geq 0$, $i \in \{1, \ldots, q\}$, be a load indicator of the i-th subtree. Denote the set of subtrees distributed to z-th computational node as $\mathcal{S}(z)$, $z \in \{1, \ldots, Z\}$. Then, the load of the z-th node is $L_z = \sum_{i:s_i \in \mathcal{S}(z)} \alpha_i$, and the maximum load across all nodes is $L = \max_z L_z$. So, the LBP problem is to find an assignment $\mathcal{S}(z)$, $z \in \{1, \ldots, Z\}$, such that L is minimized.

Based on Def. 3.2, minimizing the overall load L essentially means finding Z disjoint subsets of $\mathcal{S}$ on which computation loads L_z ($z \in \{1, \ldots, Z\}$) are close. This, however, is an NP-complete problem (Cormen et al. 2001). Thus, we adopt the *longest processing time* (LPT) strategy which has been proved to be a 4/3-approximation (Graham 1969) for load balancing. The LPT strategy for CP-tree partitioning is applied by sorting all subtrees in descending order based on their load indicators α_i, and then sequentially allocating each tree to the node that currently has the lowest estimated computational load. This process continues until all subtrees have been assigned. The performance of LBP mainly depends on how well the load indicator values (α_i) can be estimated, and below we propose two simple heuristic approaches for estimating actual computational load.

Intuitively, a subtree of CP-tree rooted with a higher support item is likely to represent patterns that are more commonly (frequently) found in a user population. Thus, such a subtree has a greater chance to be traversed for eligible patterns and, therefore, can entail more intensive calculations. Thus, our first heuristic approach is a *support-based* load indicator, where we use the support value of the subtree root item as the subtree load indicator. Alternatively, the size of a subtree is also related to traversal cost, as bigger subtrees contain more patterns. In other words, the eligible pattern discovery process may result in heavier computation load due to having to check potentially more patterns. Therefore, our second heuristic approach is a *pattern-based* load indicator, where we adopt the subtree size (i.e., number of patterns in a subtree) as the subtree load indicator.

Note that neither of the two indicators dominates the other theoretically; which one works better is context-dependent. The support-based load indicator is likely to be more suitable for relatively

Table 4: Summary statistics of datasets.

	Rating	#Users	#Items	#Ratings	Sparsity(%)	Avg.#Ratings per Item	% of Ratings from Top 20% Items	% of Ratings from Top 50% Items	Gini Coefficient
`Book-Crossing`	[1,10]	1834	2172	41,337	98.96	13	40%	65%	0.37
`Last.fm`	[1,5]	635	4100	14,175	99.46	5	53%	70%	0.57
`MovieLens`	[1,5]	943	1682	100,000	93.70	59	65%	93%	0.63

Sparsity=100(1−#Ratings/(#Users×#Items)), i.e., percentage of unknown ratings.

sparse data. In such cases, on average, C_u would contain few items, so most subtrees would have limited eligible patterns for recommending certain target items to u. This indicates that the computational cost would mainly come from multiple traversals of frequent subtrees, which would be captured by the support-based load indicator. In contrast, the pattern-based indicator is likely to perform better on denser data. In this case, on average, C_u would contain more items, and most of the subtrees are likely to be traversed. Thus, the computational cost may depend much more on the size of subtrees (the number of patterns to be checked), which would be estimated by the pattern-based load indicator. We compare the two indicators empirically in Section 4.5.

4 Experimental Results

In this section, we conduct computational experiments to evaluate CORE. We first compare CORE to two types of baselines, i.e., the pattern-based and the classic CF-based (Collaborative Filtering) methods. We then analyze the advantages of CORE for long-tail recommendation followed by a discussion on the flexibility of CORE in recommending items of different popularity levels.

4.1 Experimental Setup

Data. CORE is tested using three publicly available datasets[4] that are widely used in recommender systems research: `Book-Crossing`, `Last.fm`, and `MovieLens`. For `Book-Crossing` and `Last.fm`, we sample the data to include users with at least ten ratings to avoid extreme sparsity. Table 4 provides the summary statistics of the three data sets, including the information about the distributional inequality in item consumption. E.g., if we consider the most popular 20% of items in each dataset, in `MovieLens` such items receive over 65% of all ratings, while the numbers are significantly lower for the other two datasets, i.e., 40% and 53%, indicating a heavier-tailed nature of `Book-Crossing` and `Last.fm` datasets. The Gini Coefficient values further highlight the differences among datasets in terms of item rating frequency distributions.

[4]The datasets can be found at `http://grouplens.org/datasets/hetrec-2011/`.

Performance metrics. Even though the main motivation behind the proposed approach is addressing the long-tail recommendation challenges, recommendation accuracy is always an important performance dimension. Precision (or precision-in-top-k) is commonly used to evaluate the accuracy of top-K recommendation lists and is calculated as follows. For each user u,

$$Precision_u = \#hits_u/K, \tag{2}$$

where $\#hits_u$ is the number of items from user u's recommendation list that are also in u's test set, and K is the length of the recommendation list (by default, we use $K = 10$ in our experiments).

To evaluate the long-tail performance of recommendation algorithms, we use two metrics: (i) the average popularity ($AvgPop$) of items in the top-K recommendation list, where each item's popularity is reflected by the number of ratings it has (Yin et al. 2012, Niemann and Wolpers 2013), and (ii) the ratio of niche items ($NicheRatio$) in the top-K recommendation list. In a given dataset, an item is defined to be a *niche* item if it has fewer ratings than the average number of ratings per item in the data set. More formally, for each user u,

$$AvgPop_u = (\sum_{n=1}^{K} \#ItemRatings_n)_u/K, \qquad NicheRatio_u = \#NicheItems_u/K. \tag{3}$$

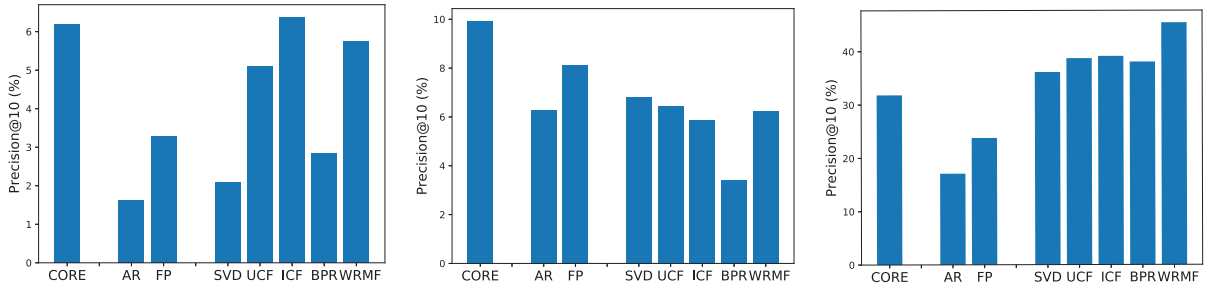
The overall performance for each metric is obtained by averaging its values over all users.

Baselines. We compare CORE with two types of baselines, i.e., pattern-based and collaborative filtering methods. The former includes the association rule-based method (AR) (Lin et al. 2002) and the frequent pattern-based method (FP) (Nakagawa and Mobasher 2003). The collaborative filtering category includes five widely adopted methods: UCF (User-based Collaborative Filtering) (Resnick et al. 1994), ICF (Item-based Collaborative Filtering) (Sarwar et al. 2001), SVD (Funk 2006), WRMF (Weighted Regularized Matrix Factorization) (Hu et al. 2008), and BPR (Bayesian Personalized Ranking) (Rendle et al. 2009). In particular, AR, FP, BPR, WRMF and CORE are designed specifically for implicit feedback data (i.e., 0/1 data that reflects only whether a user consumed or purchased an item, and not the user's explicit preference rating for that item). For unified comparison with different baselines, we first convert explicit ratings in the three data sets to consumption (i.e., 0/1) data, based on the absence/presence of user rating. We then adopt the standard split-validation method commonly used in recommender systems research to evaluate different methods; that is, we

randomly select 70% percent of the consumed items of each user for model training purposes, and use the remaining 30% of items for performance evaluation. Hyperparameters of each algorithm – e.g., the neighborhood size in UCF and ICF, the number of latent factors in SVD, WRMF, and BPR, the support and cosine thresholds for CORE, etc. – were carefully tuned using standard predictive modeling practices for best accuracy performance.

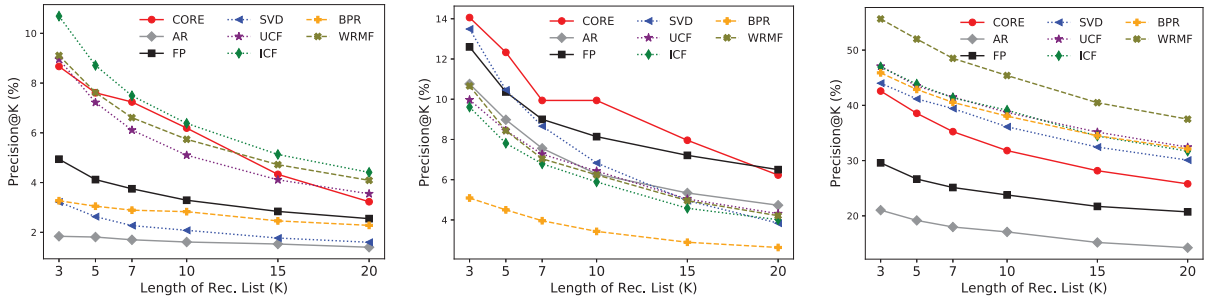
4.2 Recommendation Accuracy Performance

Fig. 3 compares CORE to different baselines in terms of precision-in-top-10 (30% of data is used for testing). In terms of accuracy comparisons with other pattern-based approaches, the results show that the precision of CORE is consistently higher than that of AR and FP across all data sets. In terms of accuracy comparisons with collaborative filtering methods, Fig. 3 shows that CORE is highly competitive on `Book-Crossing` (only very slightly behind the best baseline ICF) and `Last.fm` (best performance among all methods). On `MovieLens`, collaborative filtering baselines show performance advantages over CORE, and we will take a closer look at this later in the paper.



(a) `Book-Crossing` (b) `Last.fm` (c) `MovieLens`

Figure 3: CORE vs. baseline algorithms on accuracy (test-set ratio = 30%).

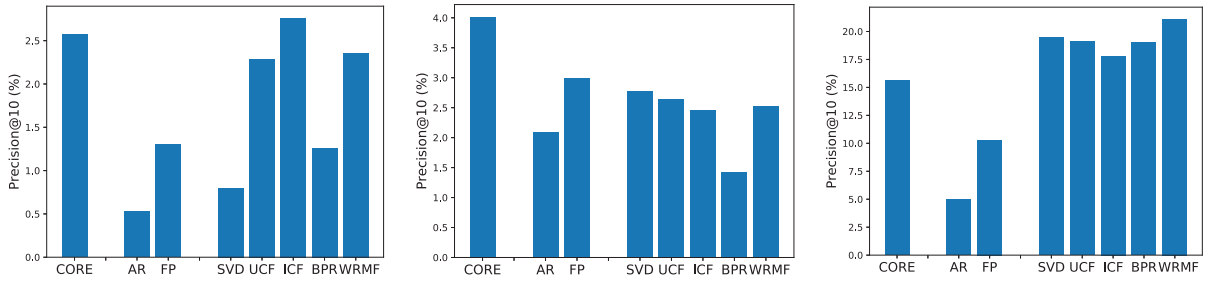


(a) `Book-Crossing` (b) `Last.fm` (c) `MovieLens`

Figure 4: CORE vs. baseline algorithms on accuracy under different K.

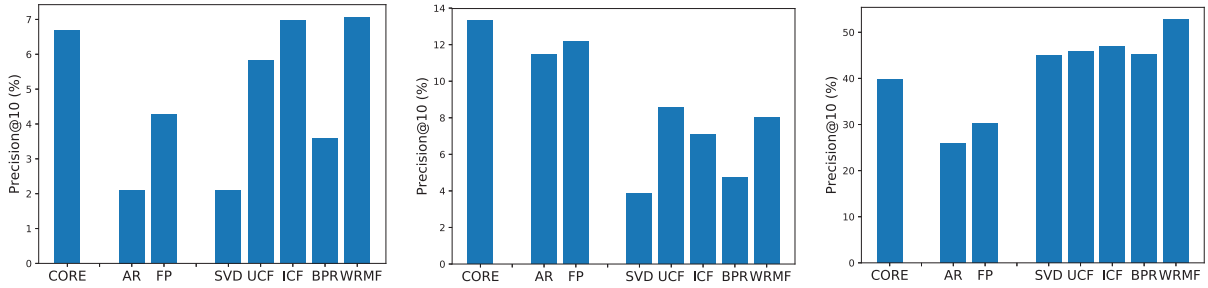
To demonstrate the robustness of the CORE performance under different settings, we provide accuracy comparisons for different lengths of the top-K recommendation list and for different splits

of training and test data. The results are shown in Figs. 4-6. Specifically, Fig. 4 shows that the relative accuracy performance of different methods remains consistent when different number of recommendations is provided. I.e., CORE remains competitive on `Book-Crossing` and demonstrates superior performance on `Last.fm`; on `MovieLens`, CORE is outperformed by collaborative filtering baselines, but performs better than pattern-based methods. Relative accuracy performance also remains consistent for different training-test data splits, as illustrated by Figs. 5 and 6, which show the precision comparisons among different methods when 90% and 50% of the data is used for model learning (the remaining 10% and 50% are used for model evaluation), respectively.



(a) `Book-Crossing` (b) `Last.fm` (c) `MovieLens`

Figure 5: CORE vs. baseline algorithms on accuracy (test-set ratio = 10%).



(a) `Book-Crossing` (b) `Last.fm` (c) `MovieLens`

Figure 6: CORE vs. baseline algorithms on accuracy (test-set ratio = 50%).

Overall, accuracy comparisons in Figs. 3-6 demonstrate highly promising performance of CORE; i.e., it dominates pattern-based methods across all data sets and is highly competitive with the widely used CF-based approaches on two heavier-tailed data sets, i.e., `Book-Crossing` and `Last.fm`.

4.3 Long-Tail Recommendation Performance

The key objective of the proposed approach is the long-tail recommendation. Therefore, in this section, we go beyond recommendation accuracy and focus on the long-tail recommendation performance evaluation. Tables 5 and 6 show the average popularity (AvgPop), i.e., average number of

Table 5: Average popularity of recommended items.

Data Set	AvgFreq	CORE	AR	FP	UCF	ICF	SVD	WRMF	BPR
`Book-Crossing`	13	43	76	101	72	45	69	65	92
`Last.fm`	5	31	89	107	61	52	34	48	96
`MovieLens`	59	135	318	383	333	338	397	301	275

AvgFreq: average number of ratings per item in a data set.

Table 6: Percentage of niche items recommended.

Data Set	AvgFreq	CORE(%)	AR(%)	FP(%)	UCF(%)	ICF(%)	SVD(%)	WRMF(%)	BPR(%)
`Book-Crossing`	13	13.50	0.05	0.04	2.10	1.38	1.81	3.20	0.40
`Last.fm`	5	16.90	1.72	0.02	2.16	4.44	5.77	9.30	0.20
`MovieLens`	59	25.35	0.00	0.00	0.02	0.20	0.00	0.11	1.41

AvgFreq: average number of ratings per item in a data set.

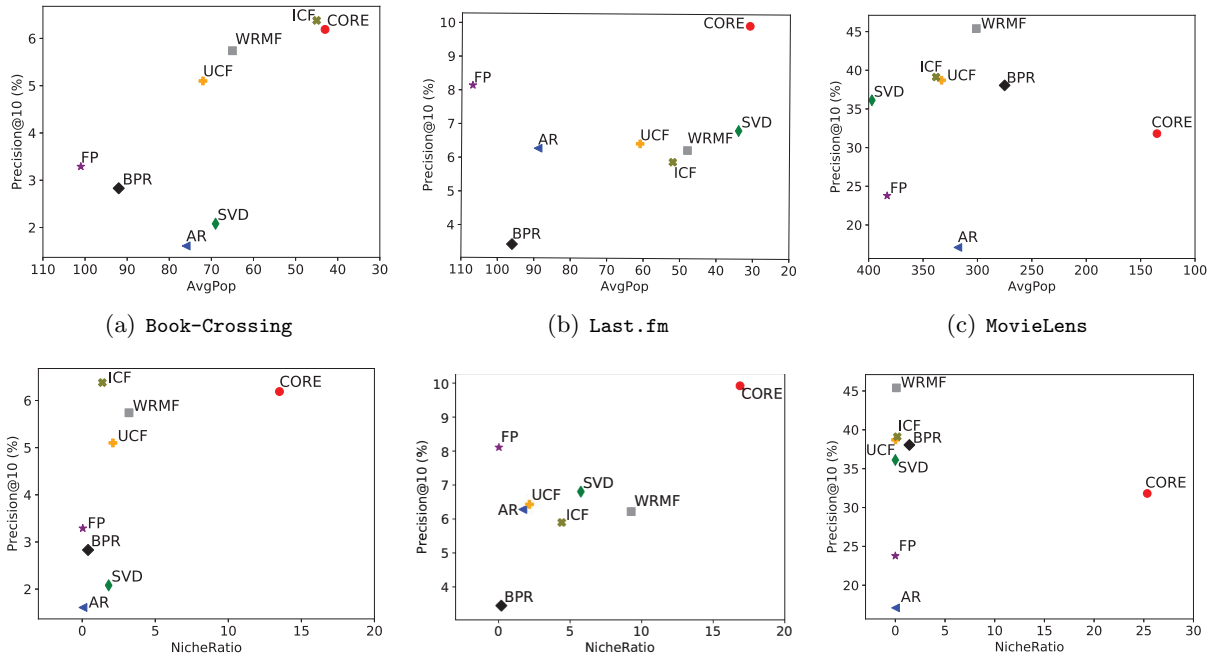
ratings, and ratio of niche items (NicheRatio) of top-10 items recommended by the same exact model configurations evaluated in Section 4.2. Again, the same 30% of data was used for testing.

The results highlight the significant advantages of CORE over baseline algorithms for long-tail recommendation. In particular, baseline algorithms tend to recommend popular items, as indicated by the average popularity metric and by the fact that, in the vast majority of settings, less than 2% of recommendations by baseline algorithms are niche items. These results are consistent with the findings of previous studies (Fleder and Hosanagar 2009) that most existing recommender systems have *popularity bias*, creating the rich-get-richer effect for popular items. In contrast, CORE is successfully able to provide recommendations of items with lower popularity and recommendations containing substantially higher percentage of niche items across all datasets.

Fig. 7 reiterates the results from Sections 4.2 and 4.3 by comparing the overall performance of different recommendation techniques in a two-dimensional (i.e., accuracy vs. long-tail recommendation) space. Specifically, Figs. 7a-7c show each method's position in the *Precision-AvgPop* performance space and, similarly, Figs. 7d-7f show *Precision-NicheRatio* comparison. Methods appearing in the top-right corner in these figures demonstrate better performance on both accuracy and long-tail recommendation. As shown in Fig. 7, for `Book-Crossing` and `Last.fm`, CORE is not only the advantageous choice in terms of the long-tail recommendation performance, but it is also an excellent overall choice based on *both* performance dimensions. For `MovieLens`, CORE demonstrates dramatic improvements in long-tail performance at the expense of some accuracy reduction with respect to CF baselines (but still significantly outperforming pattern-based baselines).

4.4 Additional Experiments

In this section, we discuss a number of additional important characteristics of the CORE approach.



(a) `Book-Crossing` (b) `Last.fm` (c) `MovieLens`

(d) `Book-Crossing` (e) `Last.fm` (f) `MovieLens`

Figure 7: CORE vs. baseline algorithms on accuracy and long-tail recommendation.

Preference for Heavier-Tailed Datasets. From the main results discussed so far, we see that traditional pattern-based and collaborative filtering methods tend to recommend items with higher popularity, while CORE is able to recommend substantially more long-tail items across all data sets. In terms of accuracy performance, on more skewed datasets (such as `MovieLens`, where small percentage of popular items are responsible for high percentage of ratings/consumptions, as could be seen in Table 4), traditional (popularity-oriented) collaborative filtering techniques tend to have some inherent advantage. However, this accuracy advantage of traditional techniques disappears on heavier-tailed data sets (such as `Book-Crossing` and `Last.fm`, with larger percentage of ratings dispersed to niche items), where CORE exhibits highly competitive accuracy performance.

We provide additional support for this finding with the following experimental evaluation, where we compare CORE with WRMF (i.e., one of the collaborative filtering baselines that demonstrates consistently good performance) on datasets with varying degrees of skewness. In particular, we take `MovieLens` data, where WRMF consistently demonstrates superior accuracy performance over other methods (including CORE), manipulate its rating distribution to achieve different levels of skewness, and investigate the changes in relative performance of WRMF vs. CORE. Specifically, we first rank all items in `MovieLens` by their support (i.e., the number of ratings), and then remove items ranking

Table 7: Descriptions of adjusted data sets.

	Rating	#Users	#Items	#Ratings	Sparsity(%)	Avg.#Ratings per item	% of Ratings from Top 20% Items	% of Ratings from Top 50% Items	Gini Coefficient
`MovieLens`	[1,5]	943	1682	10,000	93.70	59	65%	93%	0.63
`MovieLens-10%`	[1,5]	943	1512	56,960	96.00	38	57%	90%	0.57
`MovieLens-20%`	[1,5]	939	1344	35,181	97.21	26	54%	89%	0.54
`MovieLens-30%`	[1,5]	919	1175	21768	97.98	19	52%	87%	0.52
`MovieLens-40%`	[1,5]	866	1004	12,866	98.52	13	49%	84%	0.50
`MovieLens-50%`	[1,5]	756	838	7,218	98.86	9	48%	83%	0.46
`Last.fm`	[1,5]	635	4100	14,175	99.46	5	53%	70%	0.57

in top 10%, 20%, 30%, 40%, and 50%, respectively, to generate five new data sets with heavier-tailed distribution. Descriptive statistics and distributions of these new datasets are shown in Table 7, from which we can see that, the more popular items are being removed, the closer the rating distribution is to the one in `Last.fm`, one of our heavier-tailed datasets.

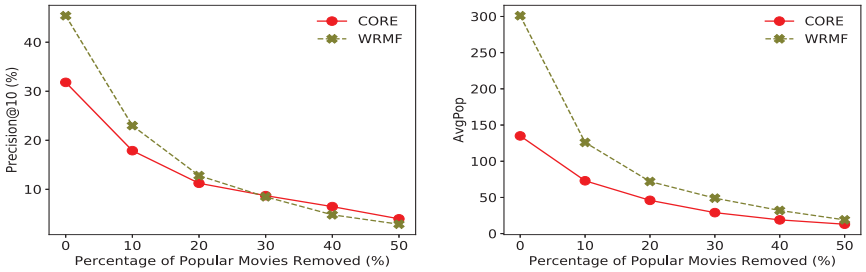


Figure 8: CORE vs. WRMF on different distributions of item popularity.

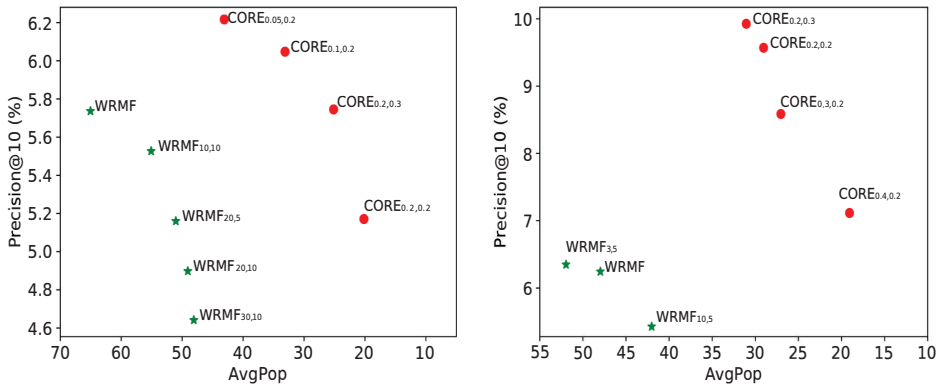
Fig. 8 shows the precision (accuracy) and AvgPop (long-tail recommendation performance) of WRMF vs. CORE on the original `MovieLens` data and five new datasets generated from it. In particular, as the rating distribution of `MovieLens` moves more toward niche items, the accuracy gap between WRMF and CORE gradually narrows. Eventually, e.g., in settings where 40% and 50% popular movies are removed, CORE actually starts to outperform WRMF in terms of accuracy. Meanwhile, the long-tail recommendation performance of CORE remains better (i.e., lower AvgPop of recommended items) than WRMF's, albeit by a smaller margin, as there are much fewer popular items for the WRMF's popularity bias to manifest itself strongly.

In summary, this provides additional evidence that, aside from demonstrating superior long-tail recommendation performance across a wide variety of datasets, on heavier-tailed datasets CORE demonstrates highly competitive performance in terms of accuracy as well.

Flexible Recommendation. An important characteristic of the proposed CORE approach is that it allows to fine-tune the popularity level of recommended items in a flexible manner. In

particular, as was mentioned in Section 3.2, the types of items that appear in the discovered cosine patterns can be easily tuned by setting cosine and/or support thresholds accordingly.

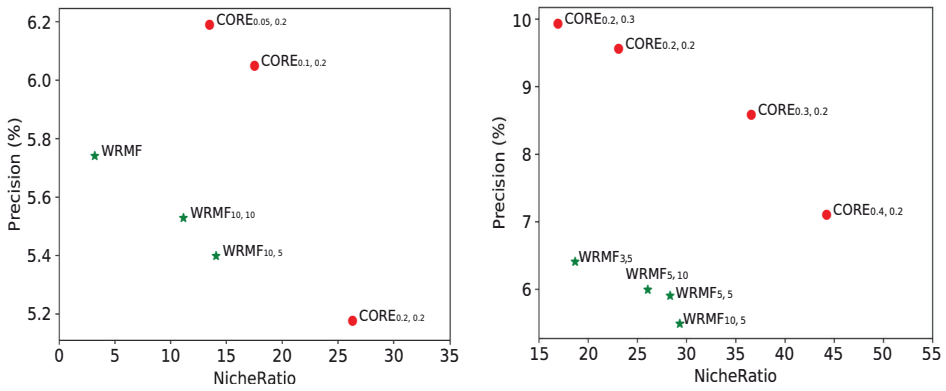
Figs. 9 and 10 show the performance of different variations of $\text{CORE}_{cos,supp}$, i.e., CORE under different cosine and support thresholds. For a given level of support, the popularity of recommended items tends to go down as the cosine threshold goes up. For `Book-Crossing` data (Fig. 9a), this can be seen from the fact that $\text{AvgPop}(\text{CORE}_{0.05,0.2}) \geq \text{AvgPop}(\text{CORE}_{0.1,0.2}) \geq \text{AvgPop}(\text{CORE}_{0.2,0.2})$. For `Last.fm` (Fig. 9b): $\text{AvgPop}(\text{CORE}_{0.2,0.2}) \geq \text{AvgPop}(\text{CORE}_{0.3,0.2}) \geq \text{AvgPop}(\text{CORE}_{0.4,0.2})$. Correspondingly, the ratio of niche recommendations tends to go up with the cosine threshold as well, as shown in Fig. 10. The intuition is that a higher cosine threshold filters out more cross-support patterns containing high-frequency (i.e., popular) items. Alternatively, for a given level of cosine, the popularity of recommended items tends to go up as the support threshold goes up. E.g., in Fig. 9a we can see $\text{AvgPop}(\text{CORE}_{0.2,0.2}) \leq \text{AvgPop}(\text{CORE}_{0.2,0.3})$ for `Book-Crossing`; in Fig. 9b we have $\text{AvgPop}(\text{CORE}_{0.2,0.2}) \leq \text{AvgPop}(\text{CORE}_{0.2,0.3})$ for `Last.fm`. The ratio of recommended niche items also correspondingly goes down. With higher support threshold, only items in comparatively more frequent patterns, i.e., more frequently consumed items, would be recommended. Not surprisingly,



(a) `Book-Crossing` (b) `Last.fm`

Figure 9: Two dimensional (*Precision-AvgPop*) performance comparison of CORE and WRMF.

fine-tuning CORE for even better long-tail recommendation performance may come at the expense of some recommendation accuracy, as Figs. 9 and 10 show. Therefore, in real-world applications, cosine and support thresholds should be set by the domain experts keeping in mind the specific application requirements, such as the desired mix of popular and niche item recommendations as well as the trade-offs between accuracy and long-tail performance.



(a) `Book-Crossing` (b) `Last.fm`

Figure 10: Two dimensional (*Precision-NicheRatio*) performance comparison of CORE and WRMF.

Comparisons with the Long-Tail-Oriented Baseline. Our main experiments demonstrated the benefits of CORE as compared to multiple popular, general-purpose baseline algorithms. The benefits are especially prominent on the heavier-tailed datasets, where CORE shows advantages not only in long-tail performance, but in accuracy performance as well. Here we conduct an additional experiment to show that CORE's performance advantages on heavier-tailed data remain even when compared to a *specialized* long-tail-oriented baseline. Specifically, we compare CORE with the long-tail recommendation strategy proposed by Park and Tuzhilin (2008). We chose this approach due to its adaptability to different existing recommendation techniques and its flexibility (somewhat similar to CORE's) for parameterizing the long-tail recommendation performance.

As with our main experiments, we hold out 30% of each users' ratings as the ground truth for evaluation purposes. Again, we chose to use the WRMF baseline in conjunction with the long-tail recommendation strategy due to WRMF's consistently good performance across different datasets. To adapt WRMF for long-tail recommendation, all items in the training data (i.e., 70% of each user's ratings) are first pre-processed by partitioning them into head (H) and tail (T) groups using different rating frequency thresholds α (depending on the overall item frequency in a given dataset). Items with the number of ratings greater than α would be in group H, the rest of the items in group T. Those in group T are further clustered into β clusters as proposed in Park and Tuzhilin (2008). The above process guarantees a more balanced item rating distribution within each group and, thus, alleviates the concern of recommendation bias towards highly popular items. In particular, we set $\alpha \in \{10, 20, 30\}$ for `Book-Crossing`, $\alpha \in \{3, 5, 10\}$ for `Last.fm`, and $\beta \in \{5, 10\}$ for both datasets. Based on the pre-processing, recommendations are first generated using WRMF within H and each of

β groups within T and then aggregated to form the final top-K recommendation list. Figs. 9 and 10 show the comparison between different variations of $\text{CORE}_{cos,supp}$ (i.e., CORE under different cosine and support thresholds, as discussed earlier) and $\text{WRMF}_{\alpha,\beta}$ (i.e., WRMF under different settings of pre-processing).

Note that, although we run a number of different CORE and WRMF variations, for clarity of visualization in Figs. 9 and 10 we display only the performance "frontiers" both for CORE and WRMF. In other words, we do not display CORE and WRMF variations where the recommendation performance is dominated by some other variations in both dimensions, i.e., *both* in recommendation accuracy and long-tail performance. First, the results verify the effectiveness of the approach proposed by Park and Tuzhilin (2008) for boosting long-tail recommendation of WRMF. Second (and, again, not surprisingly), better long-tail recommendation performance often comes at the expense of recommendation accuracy, which applies for both CORE and long-tail-oriented WRMF and CORE. And, finally and importantly, CORE still provides better performance in terms of both accuracy and long-tail recommendation, as the performance frontier of CORE dominates the one of the long-tail-oriented WRMF.

Benefits of Hybridization with CORE. As discussed earlier, CORE provides clear performance advantages on heavier-tailed data (both in long-tail and accuracy performance dimensions). However, on other kinds of datasets, such as `MovieLens`, no method strongly dominates others on both dimensions. For example, as was shown in Figs. 7c and 7f, while CORE still exhibits superior long-tail recommendation performance, it is WRMF that demonstrates best accuracy (although underperforming significantly in terms of long-tail recommendation). In such cases, it is possible to obtain advantages on both dimensions by developing a *hybrid* (or ensemble) recommender system.

As an example, we can hybridize WRMF and CORE in different ways by taking top-i recommended items from CORE and top-$(10-i)$ recommended items from WRMF and merging them into the final top-10 list. For any given $i \in \{0, 1, \ldots, 10\}$, we denote the resulting hybrid method H_i. The two-dimensional performance comparison of original WRMF (i.e., H_0), CORE (i.e., H_{10}), and all hybrid methods (i.e., H_1, H_2, ..., H_9) is shown in Fig. 11. For example, consider performance of H_5. The results indicate that, from a simple hybridization strategy of taking top-5 items from WRMF and top-5 items from CORE, H_5 is able to get nearly half of the long-tail performance benefits of CORE (over what original WRMF showed) as well as the vast majority of accuracy benefits of

WRMF (over what original CORE showed). This further highlights the practical applicability and value of CORE in a wide variety of application domains.

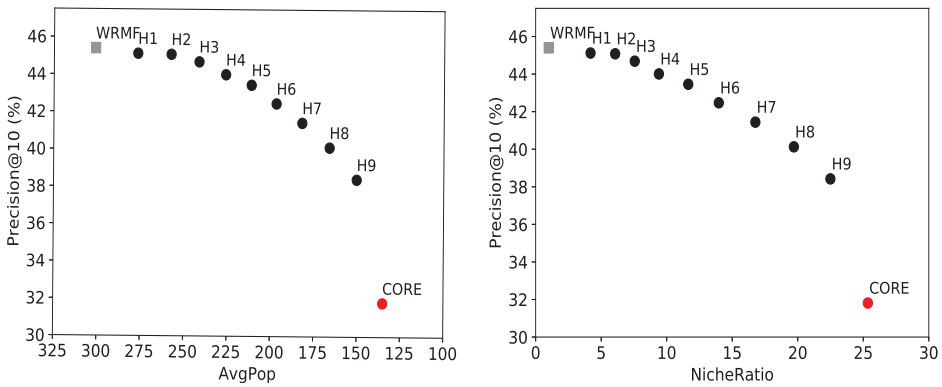


Figure 11: Performance of hybrid CORE and WRMF approaches on MovieLens data.

4.5 Scalability Demonstration: CORE for Hashtag Recommendation

In this section, we demonstrate scalability of CORE by applying it on the large-scale application of hashtag recommendation on a social media platform.

Social media platforms like Twitter or Instagram provide opportunities for instant information sharing and diffusion. However, the ease of posting encourages and facilitates rapid content generation, which can lead to information overload for the users. To deal with this issue, some platforms encourage their users to create and cite *hashtags*, i.e., relevant keywords or phrases (that start with hashtag symbol #) to indicate the themes of their posts. For example, a post with hashtag *#SuperBowl* can be easily associated with other Super-Bowl-related posts for future search, recommendation, or analysis. With the accumulation of huge numbers of hashtags over time, analyzing data of user-hashtag engagement and providing personalized hashtag recommendations in real time has become an important function of social media platforms, which facilitates discovery of relevant content and encourages further engagement.

To build and evaluate CORE for hashtag recommendation, we collected data on user engagement with different hashtags from Sina Weibo, a Twitter-like platform in China. Specifically, we collected all tweets that were posted by users over a period of one month and extracted all hashtags that were ever used. This resulted in a dataset containing 172,981,649 observations (user-hashtag interactions) from 1,629,504 users on 46,281 different hashtags (with extreme sparsity of 99.77%) and over half

Table 8: Straightforward vs. CP-tree based recommendation efficiency.

	Total	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9
Size	1,629,504	477,811	170,369	166,566	112,961	23,539	31,388	28,300	61,494	376,222	180,854
T_{CORE} (s)	2,069,134	872,005	183,828	136,917	313,354	74,713	69,430	143,906	79,389	82,016	113,576
T_{CORE+} (s)	21,049	5,439	2,758	3,500	1,219	366	229	894	270	1,036	5,338
T_{CORE++} (s)	2390	590	304	393	124	40	27	103	36	188	585
T_{CORE}/T_{CORE+}	98	160	67	39	257	204	303	161	294	79	21
pattern-based LBP T_{CORE+}/T_{CORE++}	8.8	9.2	9.1	8.9	9.8	9.2	8.4	8.7	7.5	5.5	9.1
support-based LBP T_{CORE+}/T_{CORE++}	8.4	8.3	8.9	8.7	8.7	8.4	8.3	9.3	6.8	6.2	8.5

billion tweets posted by those users. For each user, 70% of all the hashtags used were randomly chosen as the training set and the remaining 30% as the test set.

We had two different implementations of CORE for this recommendation task: (i) we applied CORE directly on the entire user population, and (ii) we partitioned the users into several more homogeneous sub-populations and applied CORE separately on each sub-population. The latter approach is a popular practice in large-scale recommender systems that can often result in better recommendations due to the use of data from more relevant user population (i.e., population with similar interests, tastes, and behavior). Clustering of users was done by first applying LDA (Blei et al. 2003), a well-known topic modeling technique, to all tweets to discover a set of different topics users discussed. Each user would then be represented as a probability distribution over discovered topics (i.e., their preferences to different topics). Based on users' topic preferences, K-means clustering was used to segment users into 10 different clusters, which we denote as C0, C1, ..., C9. The number of clusters was determined according to a commonly used evaluation procedure, i.e., elbow criterion with the sum-of-squared-errors metric. The size of each cluster (i.e., number of users) is shown in Table 8.

Our experiments showed a substantial accuracy gain of using CORE on partitioned user population. Specifically, applying CORE on the entire user population resulted in precision-in-top-10 of 31.8%, whereas the performance increased to 37.5% in the partitioned population setting. Therefore, we focus our discussion on the latter implementation in the remainder of this section.

We summarize the computational efficiency comparison of the straightforward (CORE), CP-tree-based (CORE+), and parallel CP-tree-based (CORE++) implementations of CORE for hashtag recommendation in Table 8. Specifically, T_{CORE}, T_{CORE+}, and T_{CORE++} denote the total amount of time (in seconds) that was needed to generate recommendations. The table also displays the *speedup* ratio of using CP-tree (i.e., $\mathrm{T}_{CORE}/T_{CORE+}$) as an indicator of efficiency gain. While time consumed varies with the cluster size and the number of patterns discovered within each cluster,

CORE+ is nearly 100 times faster on average. More specifically, on average, it takes about $1270ms$ to generate recommendations for a single user using the straightforward implementation, while it takes only about $13ms$ using CORE+. Finally, the last two rows of Table 8 demonstrate the additional efficiency gain due to parallelization, by comparing runtimes of CORE+ and CORE++, where CORE++ is implemented as parallel CORE+ with 12 computational nodes. In summary, CORE++ is 8-9 times faster than CORE+, and it takes only about $1.5ms$ on average for CORE++ to generate recommendations for a single user. Note that, for CORE++, two different load-balanced partitioning (LBP) strategies (as proposed in Section 3.4) were compared, and pattern-based load indicator yielded slightly better speedup than the support-based one. These scalability illustrations further emphasize the practicality and real-time capabilities of CORE.

5 Conclusions

With the increasing adoption and growth of digital content provision, recommender systems have become an indispensable component of various online platforms, as they help users to find relevant products or services from a vast array of choices more efficiently via personalized recommendations. Such systems have been reported to have significant impact on product sales and users' consumption behaviors. In particular, many traditional recommendation approaches (e.g., collaborative filtering approaches) exhibit substantial *popularity bias*, i.e., recommendations tend to direct users' attention largely towards popular products. At the same time, it is widely acknowledged that long-tail recommendations can also be valuable, both for consumers and providers, as they better satisfy heterogeneous consumer needs and can lead to increases in demand, engagement, and loyalty. For certain business models (e.g., for streaming service platforms), recommending more niche content to users could also reduce content licensing costs and, thus, lead to higher provider surplus.

However, due to the highly skewed distribution of consumed items and the fact that user preferences for more idiosyncratic and less popular items are harder to predict, accurate recommendation of long-tail items remains a significant challenge. In this paper, we propose CORE, a cosine-pattern-based technique, for effective long-tail recommendation. The proposed approach has two key components. First, it uses a special type of item co-occurrence patterns, called cosine patterns, that are mined from consumers' consumption histories and, as discussed in the paper, turn out to be highly advantageous for recommendation purposes, especially for long-tail, niche items. Second, it generates

personalized recommendations by matching each user's consumption history against the discovered patterns. To ensure scalability of the proposed approach, we design a CP-tree structure for efficient recommendation generation (CORE+) and can further employ a parallel recommendation framework (CORE++) to facilitate real-time recommendation.

In our experimental studies, we observe that cosine patterns indeed demonstrate the advantages of discovering more cohesive relationships among items, including niche items. The proposed cosine-pattern-based approach (CORE) is tested on three public datasets from different application domains, and we compare it to two types of baseline algorithms – pattern-based and collaborative-filtering-based – in terms of accuracy and long-tail recommendation performance. The results show that CORE dominates all baselines in terms of long-tail recommendation, while being highly competitive in terms of recommendation accuracy. In particular, on heavier-tailed datasets, CORE consistently demonstrates accuracy performance that on par with the highest-performing baselines. On other datasets, in addition to its superior long-tail performance, CORE offers straightforward "hybridization" opportunities for combining it with traditional top-accuracy baselines to achieve combined benefits in both accuracy and long-tail performance. Finally, in addition to its high explainability, which is common to most pattern-based recommendation approaches, CORE demonstrates high flexibility, which provides the system designers with the ability to fine-tune the system (i.e., using different thresholds for support and cosine metrics) towards the desired popularity of recommended items, as well as high scalability, which enables to facilitate real-time recommendations for a given user (i.e., in a matter of milliseconds) in large-scale recommendation applications.

This study also provides several directions for future research. One such direction would be to further our understanding regarding the impact of dataset characteristics on the performance of cosine-pattern-based recommendation method. As shown in the paper, skewness in item popularity and consumption has an impact on the cosine patterns that are discovered and, hence, on the performance of the proposed algorithm. Developing a deeper mathematical understanding of the role that specific dataset characteristics play in the cosine-pattern-based recommendation performance would allow to further improve the effectiveness of pattern-based recommendation algorithms. Another promising research direction would be to move beyond pattern-based methods and to use advantages of the cosine metric more directly in the recommendation generation process, e.g., perhaps as part of a rating-prediction or learning-to-rank approach based on supervised machine learning method-

ologies. More specifically, the current approach uses the cosine metric to learn item associations (patterns) first, and then generates recommendations based on patterns. Bypassing the intermediate step of pattern generation and designing more direct methods of using cosine information may lead to additional performance benefits. And, finally, conducting user studies to obtain further insights on users' interactions with (and acceptance of) cosine-based recommender systems constitutes another interesting direction for future work.

References

Adamopoulos P, Tuzhilin A (2015) On unexpectedness in recommender systems: Or how to better expect the unexpected. *ACM Transactions on Intelligent Systems and Technology* 5(4):54:1–54:32.

Adomavicius G, Kwon Y (2012) Improving aggregate recommendation diversity using ranking-based techniques. *IEEE Transactions on Knowledge and Data Engineering* 24(5):896–911.

Adomavicius G, Tuzhilin A (2005) Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* 17(6):734–749.

Agrawal R, Imielinski T, Swami AN (1993) Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data.*, 207–216.

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. *Proceedings of the 20th International Conference on Management of Data, VLDB*, 487–499.

Alshammari G, Jorro-Aragoneses JL, Kapetanakis S, Petridis M, Recio-García JA, Díaz-Agudo B (2017) A hybrid cbr approach for the long tail problem in recommender systems. *Proceedings of International Conference on Case-Based Reasoning*, 35–45 (Springer).

Anderson C (2006) *The long tail: Why the future of business is selling less of more* (Hachette Books).

Baumol WJ, Ide EA (1956) Variety in retailing. *Management Science* 3(1):93–101.

Blei DM, Ng AY, Jordan MI (2003) Latent Dirichlet Allocation. *Journal of Machine Learning Research* 3:993–1022.

Brin S, Motwani R, Silverstein C (1997) Beyond market baskets: Generalizing association rules to correlations. *Proceedings ACM SIGMOD International Conference on Management of Data*, 265–276.

Brynjolfsson E, Hu Y, Simester D (2011) Goodbye pareto principle, hello long tail: The effect of search costs on the concentration of product sales. *Management Science* 57(8):1373–1386.

Brynjolfsson E, Hu YJ, Smith MD (2006) From niches to riches: Anatomy of the long tail. *Sloan Management Review* 47(4):67–71.

Buskirk EV (2016) The most streamed music from spotify discover weekly. URL `https://insights.spotify.com/no/2016/07/07/top-music-discover-weekly/`.

Castells P, Hurley NJ, Vargas S (2015) Novelty and diversity in recommender systems. *Recommender Systems Handbook*, 881–918 (Springer).

Ceglar A, Roddick JF (2006) Association mining. *ACM Computing Surveys* 38(2):5.

Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) *Introduction to Algorithms* (MIT press).

Craw S, Horsburgh B, Massie S (2015) Music recommendation: audio neighbourhoods to discover music in the long tail. *Proceedings of International Conference on Case-Based Reasoning*, 73–87 (Springer).

Davidson J, Liebald B, Liu J, Nandy P, Vleet TV, Gargi U, Gupta S, He Y, Lambert M, Livingston B, Sampath D (2010) The youtube video recommendation system. *Proceedings of the 2010 ACM Conference on Recommender Systems*, 293–296.

Farquhar PH, Rao VR (1976) A balance model for evaluating subsets of multiattributed items. *Management Science* 22(5):528–539.

Fleder D, Hosanagar K (2009) Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity. *Management Science* 55(5):697–712.

Fleder DM, Hosanagar K (2007) Recommender systems and their impact on sales diversity. *Proceedings of the 8th ACM Conference on Electronic Commerce*, 192–199 (ACM).

Funk S (2006) Netflix update: Try this at home. URL `http://sifter.org/~simon/journal/20061211.html`.

Gama GMC, Meira W, Carvalho ML, Guedes DO, Almeida VA (2001) Resource placement in distributed e-commerce servers. *Proceedings of IEEE Global Telecommunications Conference*, volume 3, 1677–1682.

Ghoshal A, Sarkar S (2014) Association rules for recommendations with multiple items. *INFORMS Journal on Computing* 26(3):433–448.

Goldstein DG, Goldstein DC (2006) Profiting from the long tail. *Harvard Business Review* 84(6):24–28.

Graham RL (1969) Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics* 17(2):416–429.

Hamedani EM, Kaedi M (2019) Recommending the long tail items through personalized diversification. *Knowledge-Based Systems* 164:348–357.

Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 1–12.

Hart MA (2007) The long tail: Why the future of business is selling less of more by chris anderson. *Journal of Product Innovation Management* 24(3):274–276.

Hosanagar K, Fleder D, Lee D, Buja A (2013) Will the global village fracture into tribes? Recommender systems and their effects on consumer fragmentation. *Management Science* 60(4):805–823.

Hu Y, Koren Y, Volinsky C (2008) Collaborative filtering for implicit feedback datasets. *Proceedings of IEEE International Conference on Data Mining*, 263–272.

Hurley NJ (2013) Personalised ranking with diversity. *Proceedings of the 7th ACM Conference on Recommender Systems*, 379–382 (ACM).

Jannach D, Lerche L, Gedikli F, Bonnin G (2013) What recommenders recommend–an analysis of accuracy, popularity, and sales diversity effects. *Proceedings of International Conference on User Modeling, Adaptation, and Personalization*, 25–37 (Springer).

Kazienko P (2009) Mining indirect association rules for web recommendation. *International Journal of Applied Mathematics and Computer Science* 19(1):165–186.

Kekre S, Srinivasan K (1990) Broader product line: a necessity to achieve success? *Management Science* 36(10):1216–1232.

Koren Y, Bell R, Volinsky C (2009) Matrix factorization techniques for recommender systems. *Computer* 42(8):30–37.

Levy M, Bosteels K (2010) Music recommendation and the long tail. *Proceedings of the Workshop on Music Recommendation and Discovery*, 55–58.

Lin W, Alvarez SA, Ruiz C (2002) Efficient adaptive-support association rule mining for recommender systems. *Data Mining and Knowledge Discovery* 6(1):83–105.

Mobasher B, Dai H, Luo T, Nakagawa M (2001) Effective personalization based on association rule discovery from web usage data. *Proceedings of the 3rd International Workshop on Web Information and Data Management*, 9–15 (ACM).

Murakami T, Mori K, Orihara R (2007) Metrics for evaluating the serendipity of recommendation lists. *Proceedings of Annual Conference of the Japanese Society for Artificial Intelligence*, 40–46 (Springer).

Nakagawa M, Mobasher B (2003) A hybrid web personalization model based on site connectivity. *Proceedings of WebKDD*, 59–70.

Niemann K, Wolpers M (2013) A new collaborative filtering approach for increasing the aggregate diversity of recommender systems. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 955–963 (ACM).

Paraschakis D, Nilsson BJ, Holländer J (2015) Comparative evaluation of top-n recommenders in e-commerce: An industrial perspective. *2015 IEEE 14th International Conference on Machine Learning and Applications*, 1024–1031.

Park YJ (2013) The adaptive clustering method for the long tail problem of recommender systems. *IEEE Transactions on Knowledge and Data Engineering* 25(8):1904–1915.

Park YJ, Tuzhilin A (2008) The long tail of recommender systems and how to leverage it. *Proceedings of the 2008 ACM Conference on Recommender Systems*, 11–18 (ACM).

Pathak B, Garfinkel R, Gopal RD, Venkatesan R, Yin F (2010) Empirical analysis of the impact of recommender systems on sales. *Journal of Management Information Systems* 27(2):159–188.

Pessemier EA (1978) Stochastic properties of changing preferences. *The American Economic Review* 68(2):380–385.

Rendle S, Freudenthaler C, Gantner Z, Schmidt-Thieme L (2009) BPR: Bayesian personalized ranking from implicit feedback. *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, 452–461 (AUAI Press).

Resnick P, Iacovou N, Suchak M, Bergstrom P, Riedl J (1994) Grouplens: an open architecture for collaborative filtering of netnews. *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, 175–186 (ACM).

Ribeiro MT, Ziviani N, Moura ESD, Hata I, Lacerda A, Veloso A (2015) Multiobjective pareto-efficient approaches for recommender systems. *ACM Transactions on Intelligent Systems and Technology* 5(4):53:1–53:20.

Ricci F, Rokach L, Shapira B, Kantor PB (2015) *Recommender Systems Handbook* (Springer).

Sarwar B, Karypis G, Konstan J, Riedl J (2001) Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th International Conference on World Wide Web*, 285–295 (ACM).

Sarwar B, Karypis G, Konstan J, Riedl J, et al. (2000) Analysis of recommendation algorithms for e-commerce. *EC*, 158–167.

Shi L (2013) Trading-off among accuracy, similarity, diversity, and long-tail: a graph-based recommendation approach. *Proceedings of the 7th ACM Conference on Recommender Systems*, 57–64 (ACM).

Su R, Yin L, Chen K, Yu Y (2013) Set-oriented personalized ranking for diversified top-n recommendation. *Proceedings of the 7th ACM Conference on Recommender Systems*, 415–418 (ACM).

Tan PN, Kumar V, Srivastava J (2002) Selecting the right interestingness measure for association patterns. *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 32–41 (ACM).

Tan PN, Steinbach M, Kumar V (2006) *Introduction to Data Mining*.

Tan TF, Netessine S (2009) Is Tom Cruise threatened? Using netflix prize data to examine the long tail of electronic commerce, Working paper, available at http://knowledge.wharton.upenn.edu/wp-content/uploads/2013/09/13612.pdf.

Taramigkou M, Bothos E, Christidis K, Apostolou D, Mentzas G (2013) Escape the bubble: Guided exploration of music preferences for serendipity and novelty. *Proceedings of the 7th ACM Conference on Recommender Systems*, 335–338 (ACM).

Wickramaratna K, Kubat M, Premaratne K (2009) Predicting missing items in shopping carts. *IEEE Transactions on Knowledge and Data Engineering* 21(7):985–998.

Wu J, Zhu S, Liu H, Xia G (2012) Cosine interesting pattern discovery. *Information Sciences* 184(1):176–195.

Wu Z, Cao J, Wu J, Wang Y, Liu C (2014) Detecting genuine communities from large-scale social networks: a pattern-based method. *The Computer Journal* 57(9):1343–1357.

Xiong H, Tan PN, Kumar V (2006) Hyperclique pattern discovery. *Data Mining and Knowledge Discovery* 13(2):219–242.

Yin H, Cui B, Li J, Yao J, Chen C (2012) Challenging the long tail recommendation. *Proceedings of the VLDB Endowment* 5(9):896–907.

Zaíane OR (2002) Building a recommender agent for e-learning systems. *Proceedings of International Conference on Computers in Education*, 55–59 (IEEE).

Zhang M (2009) Enhancing diversity in top-n recommendation. *Proceedings of the 3rd ACM Conference on Recommender Systems*, 397–400 (ACM).

Zhang M, Hurley N (2008) Avoiding monotony: improving the diversity of recommendation lists. *Proceedings of the 2008 ACM Conference on Recommender Systems*, 123–130 (ACM).

Zhang M, Hurley N (2009) Novel item recommendation by user profile partitioning. *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, 508–515 (IEEE Computer Society).

Zhang YC, Séaghdha DÓ, Quercia D, Jambor T (2012) Auralist: introducing serendipity into music recommendation. *Proceedings of the 5th ACM International Conference on Web Search and Data Mining*, 13–22 (ACM).

Zhou T, Kuscsik Z, Liu JG, Medo M, Wakeling JR, Zhang YC (2010) Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences* 107(10):4511–4515.

Ziegler CN, McNee SM, Konstan JA, Lausen G (2005) Improving recommendation lists through topic diversification. *Proceedings of the 14th International Conference on World Wide Web*, 22–32 (ACM).