Supporting Top-K Item Exchange Recommendations in Large Online Communities

Zhan Su
School of Computing
National University of Singapore
suzhan@comp.nus.edu.sg

Anthony K. H. Tung
School of Computing
National University of Singapore
atung@comp.nus.edu.sg

Zhenjie Zhang
Advanced Digital Sciences Center
Illinois at Singapore Pte.
zhenjie@adsc.com.sg

ABSTRACT

Item exchange is becoming a popular behavior and widely supported in more and more online community systems, e.g. online games and social network web sites. Traditional manual search for possible exchange pairs is neither efficient nor effective. Automatic exchange pairing is increasingly demanding in such community systems, and potentially leading to new business opportunities. To meet the needs on item exchange in the market, each user in the system is entitled to list some items he/she no longer needs, as well as some required items he/she is seeking for. Given the values of all items, an exchange between two users is eligible if 1) they both have some unneeded items the other one wants, and 2) the exchange items from both sides are approximately of the same total value. To efficiently support exchange recommendation services, especially with frequent updates on the listed items, new data structures are proposed in this paper to maintain promising exchange pairs for each user. Extensive experiments on both synthetic and real data sets are conducted to evaluate our proposed solutions.

Categories and Subject Descriptors

H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and Retrieval—*Information filtering*; H.3.1 [**INFORMATION STORAGE AND RETRIEVAL**]: Content Analysis and Indexing—*Indexing methods*

General Terms

Algorithms, Performance

Keywords

Recommender system, Item exchange, Online community

1. INTRODUCTION

Item exchange is becoming a popular internet phenomenon and widely supported in more and more online community systems, e.g. online games and social network web sites. In *Frontier Ville*, for example, known as one of the most popular farming games with millions of players, every individual player only owns limited types of resources. To finish the tasks in the game, the players can only resort to their online neighborhood for resource exchanges [1]. Due to the lack of effective channel, most of the players are now relying on the online forum, posting the unneeded and wanted items to attract other users meeting the exchange requirements. While the items for exchange in online games are usually virtual objects, there are also some emerging web sites dedicated to the exchange services on second-hand commodities. *Shede* [4], for example, is a quick-growing internet-based product exchange platform in China, reaching millions of transactions every year. Similar web sites have also emerged in other countries, e.g. UK [3], Singapore [2] et al. However, the users on the platform are only able to find matching exchange parties by browsing or searching with keywords in the system. Despite of the huge potential value of the exchange market, there remains a huge gap between the increasing demands and the techniques supporting automatic exchange pairing.

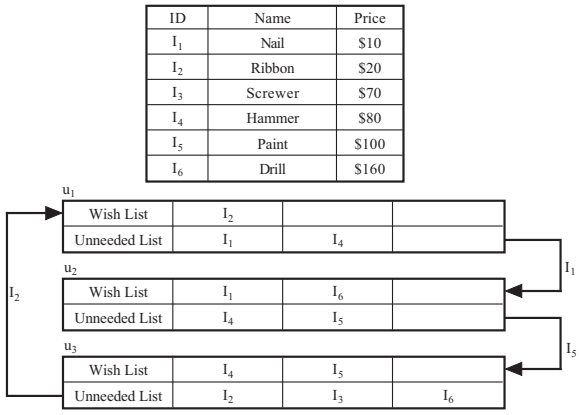


Figure 1: Example of transaction in CSEM

In this paper, we aim to bridge this gap with an effective and efficient mechanism to support automatic exchange recommendations in large online communities. Generally speaking, a group of candidate exchanges are maintained and displayed to each user in the system, suggesting the most beneficial exchanges to them. The problem of online exchange recommendation is essentially challenging in two folds. First, it is important to design a reasonable and effective exchange model, on which all users in the system are willing to follow. Second, all the recommendations must be updated in real time, to keep all users with the most recent and acceptable exchange candidates, handling the massive updates coming from every participant.

To model the behaviors and requirements of the users in the community system [9], some online exchange models have been

EDBT 2012, March 26–30, 2012, Berlin, Germany.

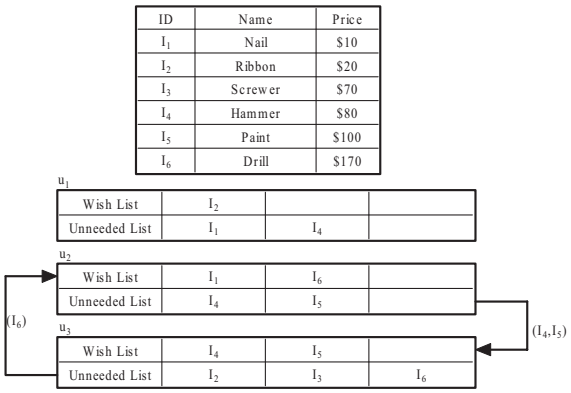


Figure 2: Example of transaction in BVEM

proposed. The recent study in [5], for example, proposed a Circular Single-item Exchange Model (CSEM). Specifically, given the users in the community, an exchange ring is eligible if there is a circle of users $\{u_1 \rightarrow u_2 \rightarrow \ldots u_m \rightarrow u_1\}$ that each user u_i in the ring receives a required item from the previous user and gives an unneeded item to the successive user. Despite of the successes of CSEM in kidney exchange problem [6], this model is not applicable in online community systems for two reasons. First, CSEM does not consider the values of the items. The exchange becomes unacceptable to some of the users in the transaction, if he/she is asked to give up valuable items and only gets some cheap items in return. Second, single-item constraint between any consecutive users in the circle limits efficiencies of online exchanges. Due to the complicated protocol of CSEM, each transaction is committed only after all involved parties agree with the deal. The expected waiting time for each transaction is too long to afford, especially in online communities. In Figure 1, we present an example to illustrate the drawbacks of CSEM. In this example, there are three users in the system, $\{u_1, u_2, u_3\}$, whose wishing items and unwanted items are listed in the the rows respectively. Based on the protocol of CSEM, one plausible exchange is a three-user circle, I_1 from u_1 to u_2, I_2 from u_3 to u_1 and I_5 from u_2 to u_3, as is shown with the arrows in Figure 1. This transaction is not satisfactory with u_2, since I_5 is worth 100$ while I_1's price is only 10$.

In this paper, we present a new exchange model, called Binary Value-based Exchange Model (BVEM). In BVEM, each exchange is run between two users in the community. An exchange is eligible, if and only if the exchanged items from both sides are approximately of the same total value. Recall the example in Figure 1, a better exchange option between u_2 and u_3 is thus shown in Figure 2. In this transaction, u_2 gives two items I_4 and I_5 at total value at $180, while u_3 gives a single item I_6 at value 170$. The difference between the exchange pair is only 10$, or 5.9% of the counterpart. This turns out to be a fair and reasonable deal for both users. On the other hand, each exchange in BVEM only involves two users, which greatly simplifies the exchange procedure. Both of the features make BVEM a practical model for online exchange, especially in highly competitive environment such as online games. To improve the flexibility and usefulness of BVEM model for online communities, we propose a new type of query, called *Top-K Exchange Recommendation*. Upon the updates on the users' item lists, the system maintains the top valued candidate exchange pairs for each user to recommend promising exchange opportunities.

Despite of the enticing advantages of top-k exchange query under BVEM on effectiveness, extensive development efforts are needed for database system, especially with large number of online users. Given a pair of two users in the community, the problem of finding the matching exchange pair with the highest total value is proven to be NP-hard, whose computational complexity is exponential to the number of items the users own. Fortunately, the size of the item lists are usually bounded by some constant number in most of the community systems, leading to acceptable computation cost on the search for the best exchange plan between two specified users. The problem tends to be more complicated if the community system is highly dynamic, with frequent insertions and deletions on the item lists of the users. To overcome these challenges on the implementation of BVEM, we propose a new data structure to index the top-k optimal exchange pairs for each user. Efficient updates on both insertions and deletions are well supported by our data structure, to maintain the candidate top-k exchange pairs.

We summarize the contributions of the paper as listed below:

1. We propose the Binary Value-based Exchange Model, capturing the requirements of online exchange behavior.

2. We design a new data structure for effective and efficient indexing on the possible exchange pairs among the users.

3. We apply optimization techniques to improve the efficiency of the proposed index structure.

4. We present extensive experimental results to prove the usefulness of our proposals.

The remainder of the paper is organized as follows. Section 2 reviews some related work on online exchange models and methods. Section 3 presents the problem definition and preliminary knowledge of our problem. Section 4 discusses the indexing structure to maintain the possible exchange pairs between two users. Section 5 extends the index structure to support more users. Section 6 evaluates our proposed solutions with synthetic data sets and Section 7 concludes this paper.

2. RELATED WORK

In this section, we review some related studies from different areas in computer science, including the kidney exchange problem in electronic commerce, the exchange game model in algorithmic game theory, and the exchange recommendation problem in database system.

The problem of kidney exchange rises from the kidney transplantation market, in which many relatives of the patients are willing to donate their kidneys but not compatible with the patients. To utilize the willing donors, a better solution is exchanging the donors among the patients [6]. With large number of patient-donor pairs, the kidney exchange problem aims to discover circles among the pairs with maximal length of L, such that the kidney of each donor is compatible to next patient on the circle. While the general problem of kidney exchange is NP-hard and difficult to find approximate solutions [8], some heuristics have been employed to find simple circles [6]. In particular, in [6] the authors proposed a linear integer programming (ILP) formulation of the kidney exchange problem. The tree search strategy with incremental formulation approach is applied to find some local optimal solution.

In computational economics, Arrow-Debreu Model is a general representation of exchange game among a group of participants with different commodities for trade [11, 12]. In this exchange game, each participant initially owns some cash as well as a combination of the commodities. Given the market prices of the commodities, the users sell unnecessary commodities and buy the some

ID	Name	Price
I_1	Nail	\$10
I_2	Ribbon	\$20
I_3	Screwer	\$70
I_4	Hammer	\$80
I_5	Paint	\$100
I_6	Drill	\$170

Time	Operation	User	Wish list	Unneeded list	Top-1	Top-2
1		u_1	I_2	I_1,I_4	--	--
		u_2	I_1,I_6	I_4,I_5	(u_2, u_3, {I_6}, {I_4, I_5})	--
		u_3	I_4,I_5	I_2,I_3,I_6	(u_3, u_2, {I_4, I_5}, {I_6})	--
2	Insert I_3 into W_1	u_1	I_2,**I_3**	I_1,I_4	**(u_1, u_3, {I_2, I_3}, {I_4})**	--
		u_2	I_1,I_6	I_4,I_5	(u_2, u_3, {I_6}, {I_4, I_5})	--
		u_3	I_4,I_5	I_2,I_3,I_6	(u_3, u_2, {I_4, I_5}, {I_6})	**(u_3, u_1, {I_4}, {I_2, I_3})**
3	Delete I_5 from U_2	u_1	I_2,I_3	I_1,I_4	(u_1, u_3, {I_2, I_3}, {I_4})	--
		u_2	I_1,I_6	I_4	--	--
		u_3	I_4,I_5	I_2,I_3,I_6	(u_3, u_1, {I_4}, {I_2, I_3})	--

Figure 3: Running Example of Top-K Exchange Pair Monitoring with $\beta = 0.8$

other commodities to optimize his utility function. The basic Arrow-Debreu Theorem [7] states that there exists a group of prices leading to a clear market, in which each user is satisfied with the final allocation. While the theorem proves the existence of the price combination with Kakutani's Theorem, it does not provide a systematic way to find the prices. In [11, 12], scientists in computer theory tried to design explicit algorithms to find the optimal prices to clear the market.

The general problem of exchange recommendation in database system is extended from the kidney exchange problem, which is closely related to our study. In [5], Abbassi and Lakshmanan proposed the Circular Single-item Exchange Model (CSEM), following the same transaction structure from kidney exchange game. CSEM is different from kidney exchange problem that each user in CSEM is allowed to take different commodities while each kidney disease patient has only one associated donor. Moreover, CSEM can be extended to some sub-models, including *Swap Exchange Model*, *Short-Cycle Exchange Model* and *Probabilistic Exchange Model*. The authors of [5] presented some algorithms to find approximate solutions to all these models with approximation factor linear to the maximal allowed cycle length k. Based on our analysis in Section 1, CSEM is only practical if the items for exchange without explicit value label and efficiency requirement. In online community space, exchanges on valued items are expected to be run with fast response time, which need better exchange model such as our proposal.

3. PROBLEM DEFINITION AND PRELIMINARIES

In the community system, we assume that there are n users $U = \{u_1, u_2, \ldots, u_n\}$, and m items $O = \{I_1, I_2, \ldots, I_m\}$. Each user u_i has two item lists, the unneeded item list L_i and the wishing item list W_i. Each item I_j is labelled with a tag v_j as its public price. Given a group of items $O' \subseteq O$, the value of the item set is the sum on the prices of all items in O', i.e. $V(O') = \sum_{I_j \in O'} v_j$. In the example for Figure 1 and Figure 2, the value of the item set $V(\{I_1, I_2, I_3\})$ =\$100 according to the price list in the figures.

In this paper, we adopt the Binary Value-based Exchange Model (BVEM) as the underlying exchange model in the community system. Given two users u_i and u_l, as well as two item sets $S_i \subseteq L_i$ and $S_l \subseteq L_l$, an exchange transaction $E = (u_i, u_l, S_i, S_l)$ represents the deal that u_i gives all items in S_i to u_l and receives S_l in return. The gain of the exchange E for user u_i is measured by the total value of the items he receives after the exchange, i.e. $G(E, u_i) = V(S_l)$. Similarly, the gain of user u_l is $G(E, u_l) = V(S_i)$. This exchange is eligible under BVEM with relaxation parameter β ($0 < \beta \leq 1$), which follows the formal definition below.

DEFINITION 1. ***Eligible Exchange Pair***
The exchange transaction $E = (u_i, u_l, S_i, S_l)$ is eligible, if it satisfies 1) Item matching condition: $S_i \subseteq W_l$ and $S_l \subseteq W_i$; and 2) Value matching condition: $\beta V(S_i) \leq V(S_l) \leq \beta^{-1} V(S_i)$.

Assuming that all users in the system are rational, each user u_i always wants to maximize his gain in the exchanges with other users. In the following, we prove the existence of a unique optimal exchange among all exchanges between u_i and u_l, maximizing both of their gains.

LEMMA 1. *For any pair of users, u_i and u_l, there exists a dominating exchange pair $E = (u_i, u_l, S_i, S_l)$ such that for any $E' = (u_i, u_l, S_i', S_l')$ the following two events can never happen: 1) $G(E', u_i) > G(E, u_i)$, or 2) $E(E', u_l) > G(E, u_l)$.*

PROOF. We prove this lemma by construction and contradiction. We order all eligible exchange pairs with non-increasing order on $G(E, u_i)$. For all exchange pairs with exactly the maximal gain for u_i, we further find the unique exchange pair $E = (u_i, u_l, S_i, S_l)$ by maximizing the gain for u_l. If E does not satisfy the condition in the lemma, there are two possible cases. In the first case, there exists an exchange pair E' that $G(E', u_i) > G(E, u_i)$. Depending on our construction method, this situation can never occur. In the second case, u_l has a better option with higher gain in $E' = (u_i, u_l, S_i', S_l')$, i.e. $G(E', u_l) = V(S_l') > G(E, u_l) = V(S_i)$. If this happens, we will show in the following that $E''(u_i, u_l, S_i', S_l)$ is also an eligible exchange pair, thus violating the construction principle of E. Based on the definition of eligible exchange pair, we know that

$$G(u_i, E') = V(S_l') \geq \beta V(S_i') = \beta G(u_l, E')$$

Since $G(u_i, E)$ is the maximal gain of u_i on any exchange pair, it is easy to verify that $V(S_l) \geq V(S_l') \geq \beta V(S_i')$. On the other hand, it can be derived that

$$V(S_l) \leq \beta^{-1} V(S_i) \leq \beta^{-1} V(S_i')$$

Combining the inequalities, we conclude $E'' = (u_i, u_l, S_i', S_l)$ is also eligible. Moreover, $G(u_i, E'') = V(S_l) = G(u_i, E)$ and $G(u_l, E'') = V(S_i') > V(S_i) = G(u_l, E)$, which also violate our construction method. This contradiction leads to the correctness of the lemma. □

The lemma suggests the existence of an optimal exchange solution between u_i and u_l for both parties, denoted by $E^*(u_i, u_l)$. However, for each user u_i, there may exist different eligible exchange pairs with different users at the same time. To suggest more promising exchange pairs to the users, we define *Top-K Exchange Pair* as below.

DEFINITION 2. ***Top-K Exchange Recommendations***
For user u_i, the top-k exchange pairs, i.e. $Top(k, i)$, includes the k most valued exchange pairs $E^*(u_i, u_l)$ with k different users.

In the definition above, each pair of user (u_i, u_l) contributes at most one exchange pair to $Top(k, i)$. It is because there is a dominating exchange plan between two users u_i and u_l. Therefore, it is less meaningful to output two different exchange suggestions between a single pair of users. The main problem we want to solve in this paper is providing an efficient mechanism to monitor top-k exchange recommendations for each user in real time.

PROBLEM 1. ***Top-K Exchange Pair Monitoring***
For each insertion or deletion on any item list L_i and W_i for user u_i, update the $Top(k, j)$ for every user u_j in the system.

Upon insertions or deletions on the item lists of user u_i, the top-k exchange pairs of u_i or other users is subject to change. Figure 3 shows an example to help understand the impact of item updates. At the initial timestamp, there is only one eligible exchange pair between u_2 and u_3, i.e. $(u_2, u_3, \{I_6\}, \{I_4, I_5\})$. The gain of u_3 in this potential exchange is 180$. At the second timestamp, assume that there is no exchange happened and a new item I_3 is inserted into u_1's wish list. The exchanging pair between u_1 and u_3 becomes eligible, as is listed in the table. The gain of u_3 from the new exchanging pair is $80, which is smaller than her gain from the previous exchange suggestion with u_2. As a result, the new exchanging pair is the second best recommendation for u_3. At time 3, I_5 is deleted from unneeded list of u_2. This breaks the existing eligible exchanging pair between u_2 and u_3, and there is no other eligible exchange pairs between them. Therefore, this exchanging pair is deleted from the recommendation list of both users. It is important to note that our system only presents the suggestions to the users, but never automatically commits these exchanges.

In the following theorem, we prove that the computation of top-1 exchange pair is difficult, even when there are only two users in the system.

THEOREM 1. *Given two users u_i and u_l, finding the optimal eligible exchange pair between u_i and u_l is NP-hard.*

PROOF. We reduce the *Load Balancing Problem* to our problem. Given a group of integers $X = \{x_1, x_2, \ldots, x_n\}$, the problem of load balancing is deciding if there exists a partition $X_1 \subset X$ and $X_2 \subseteq X$ ($X_1 \cap X_2 = \emptyset$ and $X_1 \cup X_2 = X$) that $\sum_{x_i \in X_1} x_i = \sum_{x_j \in X_2} x_j$. Load balancing problem is one of the most famous NP-hard problems [13].

Given each instance of loading balancing problem, i.e. X, we construct the item lists for u_i and u_l as follows. For each $x_j \in X$, a corresponding item I_j is constructed with value $v_j = x_j$. All these items I_j ($1 \leq j \leq n$) are inserted into the wish item list W_i for u_i and unneeded item list L_j. A new item I_{n+1} is then created with value $v_{n+1} = \sum_{x_j \in X} x_j/2$. We insert I_{n+1} into L_i and W_j. This reduction can be finished in $O(n)$ time. By setting $\beta = 0$, our problem tries to find a subset in W_i with the exact total value as I_{n+1}. If such a solution is always discovered by some algorithm in polynomial time, load balancing problem is also solvable in polynomial time. If this is the case, we will prove P=NP. □

The last theorem shows that the complexity of finding top-k exchange pair between any two users is exponential to the size of the item lists. Fortunately, the number of items owned by the users is usually limited in most of the online community systems. This partially relieves the problem of optimal exchange pairing. Therefore, the major problem for top-k exchange pair monitoring to overcome is how to effectively select some pairs of users to re-calculate the optimal exchange, when some insertion or deletion happens. In the rest of the paper, we present some data structure, which indexes the possible exchange pairs, supporting frequent updates on lists. For ease of paper reading, all of the notations are summarized in Table 1.

Notation	Description
$U = \{u_i\}$	the set of users in the community
$O = \{I_j\}$	the set of items with all users
L_i	the unneeded item list for user u_i
W_i	the wishing item list for user u_i
v_j	the value of the item I_j
$V(O')$	the value of an item set $O' \subseteq O$
S_i S_l	item subset of L_i and L_l respectively
$E(u_i, u_l, S_i, S_l)$	exchange pair between u_i and u_l
$G(E, u_i)$	the gain of u_i from exchange E
β	relaxation factor on value matching condition
$E^*(u_i, u_l)$	the optimal exchange pair between u_i and u_l
AVT	approximate value table
$AVT[m]$	mth entry in AVT
N	maximal number of items in any list
ϵ	approximation bound
$v_{\min}, v_{\max}$	minimal and maximal value of any item combination
$\mathcal{N}$	maximal number of entries in any AVT
$Top(k, i)$	Top-k exchanges list for user u_i
θ_i	minimal value of exchange pairs in $Top(k, i)$
$UL(I_j)$	set of users who have I_j in their unneeded item list
$CL(I_j)$	set of users who have I_j in their critical item set
κ	number of top results to be calculated initially
κ_i	number of top results u_i currently keep
K_i	critical item sets for user u_i

Table 1: Table of Notations

Algorithm 1 Brute-force algorithm for T1U2 exchange(L_i, W_i, L_l, W_l)

```
1: Clear optimal solutions S*
2: Generate subsets φ_L = 2^{L_i ∩ W_l} and sort on value
3: Generate subsets φ_R = 2^{L_l ∩ W_i} and sort on value
4: Set m = |φ_R|
5: for n from |φ_L| to 1 do
6:    while m > 0 and β * |φ_R[m]| > |φ_L[n]| do
7:       m = m − 1
8:    end while
9:    if φ_L[n] and φ_R[m] is an eligible exchange then
10:      S* = (u_i, u_l, φ_L[n], φ_R[m]) if V(φ_L[n]) ≥ G(S*, u_i)
         and V(φ_R[m]) ≥ G(S*, u_l)
11:   end if
12: end for
13: Return S*
```

In the following, we try to answer some common questions regarding the item exchanging model, especially on applicability and effectiveness issues:
Question 1: CSEM may find more exchanging options than BVEM does? It is true that CSEM finds more exchange candidates. However, due to the lack of value matching condition, most of the exchanges found by CSEM are meaningless in our problem domains, e.g. online games.
Question 2: Top-K exchange pairs for u_i may overlap with each other? Our BVEM only provides recommendations for exchanges. Users in the real system may decide which exchange to commit based on his own preference. An online game player, for example, is more willing to trade for a specific weapon than the others.
Question 3: What about using currency as intermediate medium

between users? Real/virtual currency is not used in many online communities, e.g. *Frontier Ville*. Even in some applications allowing direct buying/selling operation with the central system, direct exchanges are popular behavior with the users, because of the efficiency on getting highly prioritized items.

4. EXCHANGE BETWEEN TWO USERS

In this section, we focus on a special case of the exchange recommendation problem, with only two users in the system looking for the top-1 valued exchange pair between them. In the following sections, we extend our discussion to the general case with arbitrary number of users. For simplicity, we call it the *T1U2 Exchange*. Algorithmically, T1U2 exchange can be solved by an offline algorithm with exponential complexity in term of the list sizes.

The offline algorithm works as follows. It first computes the intersections between the wish list and unneeded list, i.e $W_i \cap L_l$ and $L_i \cap W_l$. Then all the subsets of the two temporary lists are enumerated. The algorithm tests every pair of the subsets to find the pairing satisfying Definition 1 and maximizing the gain of both users. Details about this algorithm is illustrated in Algorithm 1. The running time of this algorithm is exponential to the list size, i.e. $O(|S_i|2^{|S_i|}+|S_l|2^{|S_l|})$. Unfortunately, there does not exist any exact algorithm with polynomial complexity, unless P=NP. Hence it is more interesting to find some alternative solution, outputting approximate results with much better efficiency.

DEFINITION 3. ***ϵ-Approximate T1U2 Exchange for*** u_i
Assuming $E^* = (u_i, u_l, S_i, S_l)$ is the highest valued exchange pair between user u_i and u_l, an exchange pair, $E' = (u_i, u_l, S'_i, S'_l)$, is said to be ϵ-approximate for u_i if the gain is no worse than E^* by factor $1-\epsilon$, i.e. $G(E', u_i) \geq (1-\epsilon)G(E^*, u_i)$.

Different from exact top-1 exchange pairing, ϵ-approximate exchange does not possesses the similar property in Lemma 1. An ϵ-approximate exchange pair for u_i may not be ϵ-approximate for u_l. Therefore, the computation involving u_i and u_j may return different results to the users.

Inspired by the famous polynomial-time approximation algorithm on the subset sum problem [10], we design a fully polynomial-time approximation scheme(FPTAS) to calculate ϵ-approximate T1U2 exchange. Moreover, we show how to utilize the solution to design a reusable index structure to support updates.

The approximation scheme follows the similar idea in the FPTAS on subset sum problem. Generally speaking, the original brute-force algorithm spends most of the time on generating all the item combinations of $W_i \cap L_l$ and $L_i \cap W_l$. There are many redundant combinations, which share almost the same value with others. In the new algorithm, it only generates some of the combinations of the items in $W_i \cap L_j$ and $L_i \cap W_j$. These combinations are maintained in table indexed by their approximate values. The other item combinations are merged into the table when their value is similar to the existing ones. In particular, given the approximation factor ϵ, the exact value of an item set, $V(O')$, is transformed to some approximate value, $\gamma(O')$, guaranteeing that

$$V(O') \leq \gamma(O') \leq (1-\epsilon)^{-1}V(O') \quad (1)$$

To achieve this, we utilize the following rounding function $f(x)$. In the function, $v_{\max}$ and $v_{\min}$ are the maximal and minimal values of any non-empty item combination. The parameter ϵ is the error tolerance and N is the maximal number of items.

$$f(O') = \left\lceil \frac{\log v_{\min} - \log V(O')}{\log\left(1-\frac{\epsilon}{N}\right)} \right\rceil \quad (2)$$

Algorithm 2 AVT **Generation** (Item set O', Error bound ϵ , maximal value $v_{\max}$, minimal value $v_{\min}$, maximal item number N)

```
Generate an empty approximate value table AVT
Create a new entry AVT[0]
Set AVT[0].lbi = ∅
Set AVT[0].ubi = ∅
Set AVT[0].value = 0
Set AVT[0].lb = AVT[0].ub = 0
for each item I_j ∈ O' do
    for each entry AVT[m] ∈ AVT do
        Calculate M = f(AVT[m].value + v_j)
        if there is AVT[n].value = M then
            if AVT[m].lb + v_j < AVT[n].lb then
                Update AVT[n].lb and AVT[n].lbi
            end if
            if AVT[m].ub + v_j > AVT[n].ub then
                Update AVT[n].ub and AVT[n].ubi
            end if
        else
            Create a new entry AVT[n] in AVT
            AVT[n].value = M
            AVT[n].lb = AVT[m].lb + v_j
            AVT[n].ub = AVT[m].ub + v_j
            AVT[n].lbi = AVT[m].lbi ∪ {I_j}
            AVT[n].ubi = AVT[m].ubi ∪ {I_j}
        end if
    end for
end for
Return AVT
```

Intuitively, $f(O')$ is the minimal integer m that $v_{\min}\left(1-\frac{\epsilon}{N}\right)^{-m} \geq V(O')$. Since $v_{\min} \leq V(O') \leq v_{\max}$ and $f(O')$ always outputs an integer, $f(O')$ can only be a non-negative integer between 0 and $\mathcal{N} = \lceil(\log v_{\min} - \log v_{\max})/\log(1-\frac{\epsilon}{N})\rceil$. Based on this property, we implicitly merge the item combinations to $\mathcal{N}$ groups, i.e. $\{S_1, S_2, \ldots, S_{\mathcal{N}}\}$. Each group S_m contains every item combination O' with $f(O') = m$, i.e. $S_m = \{O'|f(O') = m\}$. For every item combination $O' \in S_m$, we have the common approximate value $\gamma(O')$ for O', i.e. $\gamma(O') = v_{\min}\left(1-\frac{\epsilon}{N}\right)^{-m}$, which satisfies Equation (1).

These groups are maintained in a relational table, called *Approximate Value Table* (or AVT in short). In AVT, each entry $AVT[m]$ records some statistical information of the group S_m, to facilitate the computation of ϵ-approximate T1U2 exchange. Specifically, we use $AVT[m].value$ to denote the common approximate value of all item combinations in S_m. We use $AVT[m].lb$ ($AVT[m].ub$ resp.) to denote the lower bound (upper bound resp.) of all the item combinations in S_m. We also keep the item combinations achieving the lower bound and upper bound, i.e. $AVT[m].lbi$ and $AVT[m].ubi$. In Table 2, we present an example of AVT.

To construct the AVT table, we sort all items based on their identifiers. At the beginning, the algorithm initializes the first entry $AVT[0]$ in the table. We set $AVT[0].value = AVT[0].lb = AVT[0].ub = 0$, empty $AVT[0].lbi$ and $AVT[0].ubi$ at the same time. For each item I_j in the input item set O', the algorithm iterates every existing entry $AVT[m]$ in the AVT and updates as follows. For every entry $AVT[m]$, our algorithm tries to generate a new entry $AVT[n]$ with $n = f(AVT[m].value + v_j)$. If $AVT[n]$ already exists, it tries to merge I_j into $AVT[m].lbi$ and $AVT[m].ubi$, checking if they can generate new lower and upper bound for group S_n. If $AVT[n]$ does not exist in the table, a new entry is created. The details are available in Algorithm 2.

Entry	approximate value	lb	lbi	ub	ubi	All item combinations
$AVT[1]$	2	2	$\{I_1\}$	2	$\{I_1\}$	$\{I_1\}$,$\{I_2\}$
$AVT[2]$	4	3	$\{I_3\}$	4	$\{I_1, I_2\}$	$\{I_3\}$,$\{I_1, I_2\}$
$AVT[3]$	8	5	$\{I_1, I_3\}$	7	$\{I_1, I_2, I_3\}$	$\{I_1, I_3\}$,$\{I_2, I_3\}$,$\{I_1, I_2, I_3\}$

Table 2: Example of approximate value table on a 3-item set

If we run the algorithm on a 3-item set $O' = \{I_1, I_2, I_3\}$ with item prices $v_1 = 2$, $v_2 = 2$ and $v_3 = 3$, the result AVT is presented in Table 2, with $(1 - \epsilon/N)^{-1} = 2$ and $v_{\min} = 1$. There are 7 non-empty combinations in O', including $\{I_1\}$, $\{I_2\}$, $\{I_3\}$, $\{I_1, I_2\}$, $\{I_1, I_3\}$, $\{I_2, I_3\}$ and $\{I_1, I_2, I_3\}$. After finishing the construction of the AVT table, there are only 3 entries in the table, which is much smaller than than the original number of item combinations. The information of the groups are all listed in the rows of the table. We also include the concrete item combinations in the last column for better elaboration, although AVT does not maintain them in the computation.

In the following lemma, we show that the output AVT summarizes every item combination within error bound ϵ.

LEMMA 2. *Given any item set O', for each item combination $O'' \subseteq O'$, the AVT table calculated by Algorithm 2 contains at least one entry $AVT[m]$ that*

$$V(O'') \geq (1 - \epsilon) AVT[m].value$$

$$AVT[m].lb \leq V(O'') \leq AVT[m].ub$$

PROOF. For simplicity, let $\delta = 1 - \epsilon/N$. We apply mathematical induction to that, $\forall O'' \in O'$, there is an $AVT[n]$ such that:

$$V(O'') \geq \delta^{|O''|} AVT[m].value \tag{3}$$

$$AVT[m].lb \leq V(O'') \leq AVT[m].ub \tag{4}$$

Basically, if $|O''| = 0$, namely $O'' = \emptyset$, the Equation 3 and 4 hold by giving $AVT[0]$.

Then we inductively prove the lemma. Assume that the the Equation 3 and 4 hold for all $|O'''| = k$, we are going to prove that they also hold for O'' with length $k+1$. Let $O'' = \{I_1, I_2, \ldots, I_{k+1}\}$. By the assumption, for $O''' = \{I_1, I_2, \ldots, I_k\}$, there is a $AVT[n]$ such that Equation 3 and 4 holds. According to line 9-12 in Algorithm 2, the AVT table is updated according to I_{k+1} and $AVT[n]$. Let the updated (line 11-14) or new created (line 16-21) AVT entry be $AVT[m]$. We can verify that:

$$\begin{aligned} V(O'') &= V(O'' - I_{k+1}) + v_{k+1} \\ &\geq \delta^k AVT[n].value + v_{k+1} \\ &\geq \delta^k (AVT[n].value + v_{k+1}) \\ &\geq \delta^{k+1} f(AVT[n].value + v_{k+1}) \\ &= \delta^{k+1} AVT[m].value \end{aligned}$$

$$\begin{aligned} V(O'') &= V(O'' - I_{k+1}) + v_{k+1} \\ &\geq AVT[n].lb + v_{k+1} \\ &\geq AVT[m].lb \end{aligned}$$

$$\begin{aligned} V(O'') &= V(O'' - I_{k+1}) + v_{k+1} \\ &\leq AVT[n].ub + v_{k+1} \\ &\leq AVT[m].ub \end{aligned}$$

Since $\delta^k \geq \delta^N = (1 - \epsilon/N)^N \geq 1 - \epsilon$, Lemma 2 holds. □

The size of AVT is no larger than $\mathcal{N}$. Therefore, the complexity of the AVT construction algorithm is $O(\mathcal{N}^2|O'|)$. Assuming $v_{\max}$, $v_{\min}$, ϵ and N are all known constants, the algorithm finishes in linear time with respect to the item size $|O'|$, which is supposed to be much faster than the exact algorithm if $\mathcal{N}$ is much smaller than $2^{|N|}$.

To utilize AVT in T1U2 exchange problem, we create two tables AVT_1 and AVT_2, based on $L_i \cap W_l$ and $W_i \cap L_l$ respectively. If there is an eligible exchange pair between u_i and u_l, the following lemma shows that there must also exist a pair of $AVT[m] \in AVT_1$ and $AVT[n] \in AVT_2$ with close values.

LEMMA 3. *If $E = (u_i, u_l, S_i, S_l)$ is any eligible exchange and $\epsilon \leq 1 - \beta$, there exists two entries $AVT_1[m] \in AVT_1$ and $AVT_2[n] \in AVT_2$ that*

$$\beta AVT_1[m].lb \leq AVT_2[n].ub \leq \beta^{-1} AVT_1[m].lb$$

$$\beta AVT_2[n].lb \leq AVT_1[m].ub \leq \beta^{-1} AVT_2[n].lb$$

PROOF. According to Lemma 2, we can find $AVT_1[m]$ and $AVT_2[n]$ such that $AVT_1[m].lb \leq V(S_i) \leq AVT_1[m].ub$, and $AVT_2[n].lb \leq V(S_l) \leq AVT_2[n].ub$. There could be two cases:

- $AVT_1[m].value \geq AVT_2[n].value$
- $AVT_1[m].value < AVT_2[n].value$

These two cases correspond to the two inequalities respectively. We will only prove the first case because of the symmetry.

The left side of the inequations:

$$\begin{aligned} \beta AVT_1[m].lb &\leq \beta V(S_i) \\ &\leq V(S_l) \\ &\leq AVT_2[n].ub \end{aligned}$$

The right side of the inequations:

$$\begin{aligned} AVT_2[n].ub &\leq AVT_2[n].value \\ &\leq AVT_1[m].value \\ &\leq (1 - \epsilon)^{-1} AVT_1[m].lb \\ &\leq \beta^{-1} AVT_1[m].lb \end{aligned}$$

So far the first case has been proven. The second case can be proven similarly. □

The last lemma shows that we can find candidate pairs from the approximate value tables, by testing the lower bounds and upper

Algorithm 3 Exchange Search on AVT (lists W_i, L_i, W_l, L_l)

1: Clear result set RS_i for u_i and RS_l for u_l
2: Generate AVT_1 on $W_i \cap L_l$ and AVT_2 on $L_i \cap W_l$
3: **for** each pair of entries $AVT_1[m] \in AVT_1$ and $AVT_2[n] \in AVT_2$ **do**
4: **if** $\beta \leq \frac{AVT_1[m].ub}{AVT_2[n].lb} \leq \frac{1}{\beta}$ and $\beta \leq \frac{AVT_2[n].ub}{AVT_1[m].lb} \leq \frac{1}{\beta}$ **then**
5: Generate $(u_i, u_l, AVT[m].ubi, AVT[n].lbi)$ for u_i and $(u_i, u_l, AVT[m].lbi, AVT[n].ubi)$ for u_l
6: Update RS_i and RS_l if necessary
7: **end if**
8: **end for**
9: Return RS_i to u_i and RS_l to u_l

bounds of the entries. Based on the lemma, we present algorithm 3 to show how to discover ϵ-approximate exchange pair for u_i and u_l at the same time. Note that the results for u_i and u_l may not be the same exchange pair. Given the AVT_1 on $W_i \cap L_l$ and AVT_2 on $L_i \cap W_l$, every pair of entries $AVT[m] \in AVT_1$ and $AVT[n] \in AVT_2$ are tested. If the condition in Lemma 3 is satisfied, two pairs of eligible exchange pair are generated, i.e. an exchange candidate $(u_i, u_l, AVT[m].ubi, AVT[n].lbi)$ for u_i and another exchange candidate $(u_i, u_l, AVT[m].lbi, AVT[n].ubi)$ for u_l respectively. The algorithm then tests the optimality of the two exchange pairs for u_i and u_l separately. After finding all the eligible exchange pairs, the optimal solutions are returned to u_i and u_l separately.

THEOREM 2. *Algorithm 3 outputs ϵ-approximate optimal top-k exchange pair between any two users u_i and u_l in linear time.*

PROOF. Consider the top-1 eligible exchange (u_i, u_l, S_i, S_l). By Lemma 3, we can find an upper (lower) bound item set S_i' in AVT_1, and an lower (upper, *resp.*) bound item set S_l' in AVT_2, such that they form an eligible exchange, and $V(S_i') \geq (1-\epsilon)V(S_i)$, $V(S_l') \geq (1-\epsilon)V(S_l)$. Therefore, (u_i, u_l, S_i', S_l') is an ϵ-approximate top-1 exchange pair. Since both S_i' and S_l' are lower or upper bound item sets, and Algorithm 3 compares all pairs of lower / upper bound values, S_i' and S_j' are guaranteed to be found by Algorithm 3. □

The algorithm to find approximate T1U2 is described in Algorithm 3. Since there are at most $\mathcal{N}$ entries in either table, the time complexity of Algorithm 3 is $O(\mathcal{N}^2)$. By sorting all the entries in decreasing order on approximate value and scanning entries in top-down fashion, we can easily reduce the complexity of the algorithm to $O(\mathcal{N})$.

5. GENERAL TOP-K EXCHANGE

In last section, we use the technique of approximate value table to search top-1 exchange pair between two users u_i and u_l. In real systems, however, there are usually thousands of users online at the same time. To support large community systems for exchange recommendation, we extend our discussion from two users to arbitrary number of users in this section. A straightforward solution to the problem is maintaining $|U|(|U|-1)$ approximate value tables. For each pair of users u_i and u_l, two approximate value tables AVT_{il} and AVT_{li} are constructed and maintained for item combinations in $W_i \cap L_l$ and $L_i \cap W_l$ respectively. Upon any update of the lists with user u_i, the system re-computes T1U2 between u_i and any other user u_l. $Top(k,i)$ and $Top(k,l)$ are thus updated accordingly with respect to the new optimal exchange between u_i and u_l. Unfortunately, this solution is not scalable in large online community systems on table indexing and maintenance, due to the quadratic number of tables used in this solution.

To reduce the memory space used by the index structure, we do not dynamically maintain approximate value tables between every pair of users. Instead, some lightweight index structure is kept in the system, with space consumption linear to the number of items. Given an update on some list L_i (or W_i) on user u_i, this data structure is used to find out every user u_l with potentially affected $Top(k,i)$ or $Top(k,l)$. To accomplish this, we first derive some necessary condition on top-k exchange pairs, with the concept of *Critical Item Set*.

DEFINITION 4. *Given an item list W_i of user u_i, a subset of items $O' \subseteq W_i$ form a critical item set, if $V(W_i) - V(O') < G(u_i, Top(k,i))$.*

In other words, an item set O' is critical to the wish list W_i, if the rest of the items in W_i is of total value no larger than the current optimal gain of u_i. In the following, we use K_i to denote the critical item set on W_i of u_i. Note that Definition 4 only provides an sufficient condition on critical item set. Given an item list W_i, there can be hundreds of different combinations of items satisfying the definition above. In Section 5.1, we will discuss more on how to construct a good critical item set according to some criterion.

LEMMA 4. *If $Top(k,i)$ contains an exchange pair $E = (u_i, u_l, S_i, S_l)$, S_i contains at least one item I_j in the critical item set K_i with respect to W_i.*

PROOF. Suppose that S_i does not contains any item in K_i. That is, $S_i \subset W_i - K_i$. Therefore, $V(S_i) \leq V(W_i) - V(K_i) < G(u_i, Top(k,i))$. This contradicts the condition that S_i is an top-k exchange. Therefore, S_i contains at least one item in any critical item set. □

Lemma 4 implies that the system needs to re-compute the T1U2 exchange between u_i and u_l to update $Top(k,i)$, only if u_l owns at least one critical item of u_i and vice versa. This motivates our index structure based on inverted lists on critical items. There are two inverted lists on each item, i.e. $CL(I_j)$ and $UL(I_j)$. $CL(I_j)$ consists of a list of users with I_j in his critical item set, and $UL(I_j)$ includes all users with I_j in his unneeded item list.

Generally speaking, when there is an update (insertion or deletion) on W_i of user u_i, the system retrieves a group of candidate users from the inverted lists and computes T1U2 exchange. The candidate set is $\left(\bigcup_{I_j \in W_i} UL(I_j)\right) \bigcap \left(\bigcup_{I_k \in L_i} CL(I_k)\right)$. The detailed description is given in Algorithm 4. By Lemma 4, this algorithm does not miss any necessary update on the top recommendation lists. The major cost of the candidate selection is spent on merging the inverted lists on the users. To improve the efficiency of the list merging, every inverted list is sorted on the ids of the users. In the rest of the section, we discuss details on the implementations of some more efficient pruning strategies.

5.1 Critical Item Selection

In this part of the section, we dissolve the problem on the construction of optimal critical item selection according to Algorithm 4. Given the wishing item list W_i, there are a large number of different ways to construct the critical item set K_i. Generally speaking, a good critical item set is supposed to reduce the number of candidate users tested in Algorithm 4. To accomplish this, we first derive some cost model below.

Since $UL(I_j)$ keeps the set of users owning the item I_j in their unneeded item list. Basically, we assume that $|UL(I_j)|$ is relatively small, compared to the total number of users $|U|$, i.e. $|UL(I_j)| \ll |U|$. Moreover, we further assume that $UL(I_j)$ for different items

Algorithm 4 General Top-K Update(W_i,u_i)

```
Clear the left candidate user set CU_l
for each I_j in the critical item set of W_i do
    merge UL(I_j) into CU_l
end for
Clear the right candidate user set CU_r
for each I_j ∈ L_i do
    merge CL(I_j) into CU_r
end for
for each u_l ∈ CU_l ∩ CU_r do
    Compute T1U2 between u_i and u_l
    Update Top(k,i) and Top(k,l) accordingly
end for
```

are not strongly correlated. Namely, for any two distinct items I_j and I_k, $|UL(I_j) \cap UL(I_k)| \ll |UL(I_j)|$. With this assumption, the number of candidate users to check, given the critical item set K_i, can be estimated by $\sum_{I_j \in K_i} |UL(I_j)|$.

Based on the analysis above, a good critical item set is equal to the following combinatorial problem with linear constraint.

$$\text{Minimize} : \sum_{I_j \in K_i} |UL(I_j)|$$

$$\text{s.t.} \sum_{I_j \in K_i} v_j \geq V(W_i) - G(u_i, Top(k,i))$$

That is, for an user U_i, we select a set $K_i \subset W_i$, to minimize $\sum_{I_j \in K} |UL(I_j)|$, subject to the sufficient condition $\sum_{I_j \in K} v_j \geq V(W_i) - G(u_i, Top(k,i))$ in Definition 4.

Although this problem is an NP-Complete problem, a near-optimal solution can be obtained by a simple greedy algorithm. Following such construction method, the items in W_i are sorted in decreasing order of $v_j/|UL(I_j)|$. Then the items are selected one by one in this order, until the sum of the value exceeds $V(W_i) - G(u_i, Top(k,i))$.

Table 3 shows an example of system with 5 users. The value of the items are $v_1 = 70, v_2 = 40, v_3 = 20, v_4 = 35, v_5 = 80, v_6 = 10$, and $|UL(I_1)| = 3, |UL(I_2)| = 1, |UL(I_3)| = 2, |UL(I_4)| = 1, |UL(I_5)| = 2, |UL(I_6)| = 3$. u_1 has 3 items in W_i, and the critical item set is I_1 and I_2, which has a total value of $110 > v_1 + v_2 + v_3 - G(u_1, Top(k,1)) = 70$, and sum of $UL(I_1) + UL(I_2) = 4$. Other eligible critical item sets include $\{I_1, I_3\}$ and $\{I_1, I_2, I_3\}$. By sorting the item on $v_j/UL(I_j)$, we pick up the items in order $\{I_2, I_1, I_3\}$. The final critical item set is $K_i = \{I_1, I_2\}$.

5.2 Item Insertion

When an item insertion comes, the system retrieves all candidate users with some pruning condition, and re-computes the T1U2 exchange to update the top-k recommendations.

After a new item I_j is inserted into the wish list W_i of an user u_i, some new eligible exchange pairs are generated. If there is a new eligible exchange between user u_l and u_i, u_l must own this item in its unneeded item list L_i. Otherwise, this exchange pair must be tested before. Hence the candidate user set CU is initialized with the inverted list $UL(I_j)$. Then for each user u_l in CU, the system examines if u_i owns a critical item of u_l or u_l owns a critical item of u_i. If any of these two cases happens, Algorithm 3 is invoked to find the optimal exchange pair between u_i and u_l.

We give an additional example of item insertion. In the example illustrated in Table 3, if one new item I_1 is inserted into u_2's wish list W_2, the system first retrieves the users owning I_1 in their unneeded item lists. Such users include u_3 and u_5. The system then tests if these candidate users have at least one critical item of u_2. Since u_5 does not contain any u_2's critical items $\{I_6\}$, and u_2 does not contain any u_5's critical items $\{I_4, I_6\}$ in the unneeded item list. Therefore, u_5 fails the test and u_3 will be further checked by the 2-user item exchange algorithm.

5.3 Item Deletion

When removing some I_j from W_i, the deletion operation can be done in two steps. In the first step, the system deletes all the current top-k exchanges containing the deleted item. In the second step, some re-computation is run to find new top-k exchange pairs for users with insufficient exchange recommendations.

The first step in the deletion operation is implemented with some inverted list structure, allowing the system to quickly locate all top-k exchange pairs with the deleted item I_j in W_i. Assume that the users with deleted exchange pairs are all kept in a fixing user list. Algorithm 4 is then called, for each user in the list, to fix all the top-k recommendation pairs. This implies that the deletion operation is expensive if many users are added into the fixing user list.

To optimize the system performance, we propose some optimization technique possibly reducing the number of users in the fixing user list after the deletion operation. The basic idea of the optimization is maintaining top κ exchange pairs for each user u_i, with some integer $\kappa > k$. It is straightforward to verify that $Top(k,i)$ is subset of $Top(\kappa,i)$. To utilize the expanded top exchange recommendation set, the system updates $Top(\kappa,i)$ for each insertion operation. On item deletion, if one of the exchange pair $E \in Top(\kappa,l)$ is removed due to the deletion of $I_j \in W_i$, the exchange list will not be totally re-computed immediately. Instead, the new T1U2 exchange between u_i and u_l is evaluated. If the new optimal exchange on u_i and u_l remains in $Top(\kappa,l)$, it is directly inserted back into $Top(\kappa,l)$. Otherwise, the counter decreases by one from κ to $\kappa - 1$. The complete re-computation of $Top(\kappa,l)$ is delayed until the next insertion operation on lists of u_l or there is less than k exchange pairs left with the system. We can prove that the all exchange pairs in $Top(k,i)$ must be exactly maintained by the scheme. Although it incurs more cost on insertions (because of the larger critical item set), this optimization greatly improves the overall performance of the system by cutting unnecessary re-computation of top exchange pairs.

We give an additional example of item deletion. Assume that $k = 2$ and $\kappa = 3$. At first, one user u_1 has 3 top exchanges: $E_1 = (u_1, u_3, \{I_1, I_2\}, \{I_5\})$, $E_2 = (u_1, u_5, \{I_1\}, \{I_4, I_6\}$ and $E_3 = (u_1, u_2, \{I_3\}, \{I_6\})$. If I_4 is deleted from L_1, E_2 is removed from the list, and κ_1 become 2. Suppose then I_6 is deleted, E_3 is also removed and κ_1 become 1. Then re-computing is triggered, and κ_1 is reset to 3, with the top results list re-computed.

6. EXPERIMENTS

In this section, we evaluate the algorithms we proposed in pervious sections. We adapt the real life data from B2B online market as well as generating synthetic data based on some general models.

6.1 Data Generation and Experiment Settings

6.1.1 Synthetic Dataset

The first step of synthetic data generation is creating certain number of items. Each item is assigned with a value. Values are generated according to certain distributions, including exponential and Zipf distributions. The parameters of all the distributions in investigation are provided in Table 4. The maximum value and minimum value are set at 10,000 and 10 respectively. When generating the

User	W_i	L_i	$G(u_i, Top(k,i))$	Critical Item Set
u_1	I_1, I_2, I_3	I_4, I_5, I_6	60	I_1, I_2
u_2	I_2, I_6	I_3, I_5	50	I_6
u_3	I_3, I_5	I_1, I_2, I_6	80	I_5
u_4	I_1, I_4	I_6	0	I_1, I_4
u_5	I_4, I_6	I_1, I_3	10	I_4, I_6

Table 3: Example of critical item sets of 5 users

item values, the distributions are truncated to keep all prices between 10 and 10,000.

Distribution	Density Function $p(x)$	Parameter
Exponential	$\lambda e^{-\lambda x}$	$\lambda = 1$
Zipf	$\frac{1/x^s}{\sum_{n=1}^{N}(1/n^s)}$	$s = 1, N = V_{max}$

Table 4: Parameters controlling the distributions on values

In real system, users and their items are usually strongly correlated, because of the similar tastes and behaviors. To capture the diversity and clustering properties on the users and items, we setup 5 classes to model different types of users and their popular items. Each user is randomly assigned to one of the classes with equal probability. One of the class is considered as "background class", which contains all the items. Every item is also assigned to one of the other four classes with equal probability. There is an upper limit on the maximum number of items in each list N. An item list, e.g. wish list W_i or unneeded list L_i, is full if the number of items reaches the limitation. In our experiments, to test the scalability of the system, we try to keep the item list as full as possible.

After setting the parameters and assigning users and items to the classes, the synthetic data are generated with a sequence of item updates. The generation of updates consists of two phases. The first phase is the warm-up phase. The objective of this phase is to fill each user's wish and unneeded lists, thereby with more insertions than deletions. After the lists are almost full, the simulation starts the second phase. In the second phase, insertions and deletions take place with identical frequency, leading to relatively stable system workload.

In the first phase, when generating a new update, our simulation randomly selects a user with equal probability. The generator then chooses one of the wish list or the unneeded list. If the target list is not full, an insertion operation is taken. Otherwise, the generator randomly deletes one of the item in the target list. During insertion, the selection on the inserting item depends on the user's class as well as the items' class. The generator picks up a random number to decide if the item is from the same class of the user ($4/7$ probability), the "background" class ($2/7$ probability) or the other three classes ($1/7$ probability, and $1/21$ for each class). It then uniformly chooses an item from the specific class. During deletion, one item is chosen from the list with equal probability. The selection of the deleting item does not take class information into account.

In the second phase, similar to the first phase, one item list from the chosen user is selected with equal probability. If the selected item list is empty, an insertion to the item list is run. If the item list is neither full nor empty, the generator makes a randomized decision: it generates an insertion with probability 0.6, or a deletion with probability 0.4. The probabilities are able to keep all lists almost full in the second phase.

The number of updates generated in the first phase is $N * |U|$, where $|U|$ is the number of users and N is the maximal number of items in any list. The number of updates generated in the second phase is no less than $2 * N * |U|$. The performance tends to turn stable after a series of updates in the second phase.

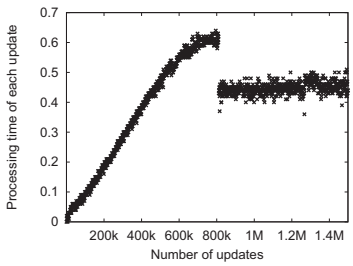


Figure 4: Average update response time over time

In Figure 4, we present the evolution of average update response time during our simulation. In the first phase of the simulation, the response time increases quickly. After transiting to the second phase, the performance tends to be stable. All our experimental results are collected in the second phase of the simulation.

The Figure 5 illustrates the distribution of the item after a period of running and the system performance has been stabilized. The amount of users in the system is 30,000 and the length of item list is limited to 15. Figure 5(a) represents the distribution of item length of each user. As we can see in the figure, the majority of users have a near-full item list. More than 80% users' item lists are of length 13, 14 or 15. Figure 5(b) illustrates the distribution on total value of each user's item list. As shown in the figure, the total value is concentrated around 15k 20k. Figure 5(c) shows the distribution on the length of the item list intersections, which is the number of common items between two users. It can be seen that users tend to have very small number of intersections. In most of the cases, it is no more than 5 items. The same trend can be seen in Figure 5(d), which plots the distribution of intersection value between users. Among all $|U|^2$ pairs of users, only a several hundred user pairs share items with more than 20k total value.

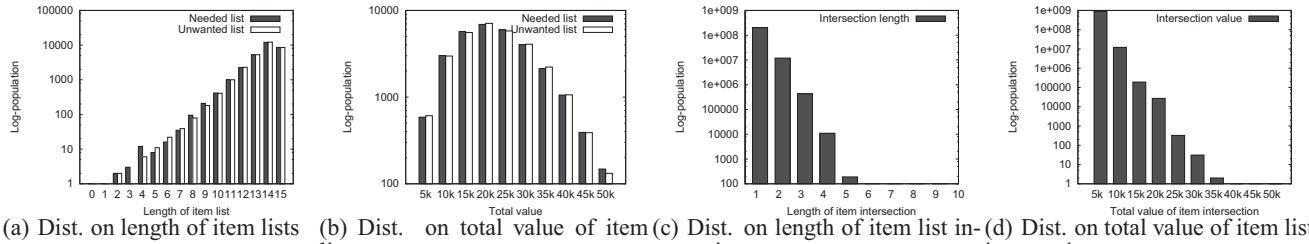
Table 5 summarizes the parameters tested in our experiments. Their default values are in bold font.

Parameter	Varying Range
Number of users	$10k, 20k,$ **30k**$, 40k, 50k$
β	$0.7, 0.75,$ **0.8**$, 0.85, 0.9, 0.95$
Length of item list	$10, 15,$ **20**$, 25, 30$
κ	$15, 25,$ **35**$, 45, 55, 65, 75$
k	$1, 3,$ **5**$, 7, 9, 11$
Number of items	$300, 600, 900, 1200,$ **1500**
ϵ	$1 - \beta$

Table 5: Varying parameters in synthetic data set

6.1.2 *Real Dataset*

It is difficult to find real exchanging data from large online communities. To get a better understanding on our method with real



(a) Dist. on length of item lists (b) Dist. on total value of item lists (c) Dist. on length of item list intersections (d) Dist. on total value of item list intersections

Figure 5: Distribution on length and total value of user item lists and intersections

world applications, we crawl some transaction data from eBay.com, which is a famous C2C online market system.

Our crawler records historical transactions with certain users in consecutive 90 days. Afterwards, all the users participating in these transactions are crawled in the same manner. In total we have crawled 34,191 users, 452,774 item records and 1,094,152 transaction records. We associate a user's wish (unneeded) list with all the item that he/she buys (sells).

As an online market is different from an exchanging market, we pre-process the data in order to make it suitable to test our system. We find that there are large number of duplicated or highly similar items. In order to reduce the duplication and increase the user item list overlapping, highly similar items are merged together. Some items and users are discarded to make sure that every user has non-empty item list. After the pre-processing, the final result data contains 2,458 users and 2,769 items.

To test our system performance under various number of users, we re-scale the data to generate data set of various size. To scale up the data, we randomly duplicate existing users until reaching the desired size. The duplicated user associates with the same set of items. To scale down the data, we randomly remove users.

We generate continuous updates according to the transactions we have crawled. We associate an item with a user's wish (unneeded) list, if this user have bought (sold) this item. To generate update operations, we randomly choose a user, an updating type (insertion/deltetion), an item list (wish/unneeded) and an item associated with this list.

The length of an item list at any moment is limited within 15. A list with 15 items are considered as full. The reason to set a fixed limitation is that our crawled transactions span 90 days. These items are not listed at the same time. At any moment, only a small number of items are listed. Therefore, we set this fixed limitation to control the number of items simultaneously listed in an item list.

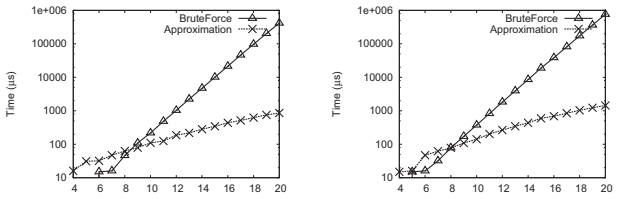
Table 6 summarizes the parameters tested in our real data experiments. Their default values are in bold font.

Parameter	Varying Range
Number of users	$0.5k, 1.5k,$ **2.5k**$, 3.5k, 4.5k$
β	$0.7, 0.75,$ **0.8**$, 0.85, 0.9, 0.95$
κ	$15, 25,$ **35**$, 45, 55, 65, 75$
k	$1, 3,$ **5**$, 7, 9, 11$
ϵ	$1 - \beta$

Table 6: Varying parameters in real data set

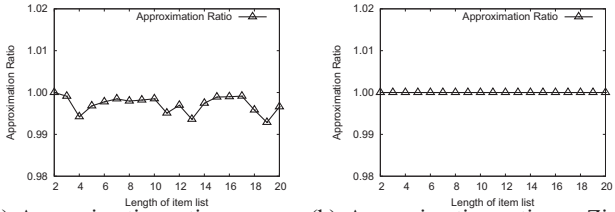
6.2 Experiments on T1U2 Exchange

In Section 4 we propose Algorithm 3, which is an approximation algorithm for finding T2U1 exchange. In this section, we evaluate its performance, including the running time and the approximation ratio. Also we use the brute force algorithm as straw-man. We test both algorithms on exponential and Zipf distribution. Detailed density functions and parameters of them are as shown in 4.



(a) Running time on exponential price distribution (b) Running time on Zipf price distribution

Figure 6: Impact of varying item list length on running time

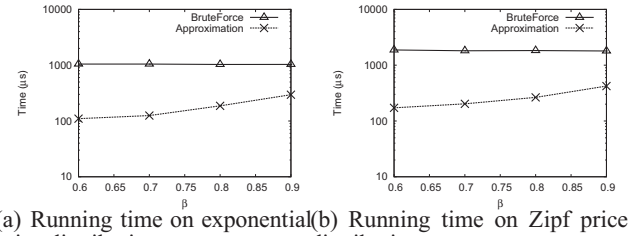


(a) Approximation ratio on exponential price distribution (b) Approximation ratio on Zipf price distribution

Figure 7: Impact of varying item list length on approximation

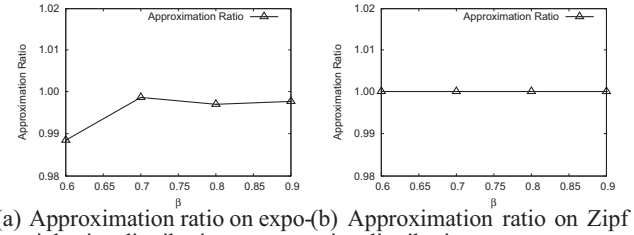
Figure 6 and 7 present the performance of both algorithms under different lengthes of item list. We fix both β and $1 - \epsilon$ to 0.8, and generate two item lists of equal length, as $W_i \cap L_l$ and $L_i \cap W_l$. Figure 6 shows the running time of both algorithms. As the plots imply, when the lengths of the item lists are less than 8, approximation scheme is not as good as brute-force algorithm, because approximation method spends too much time on index construction. However, with the size of the item set grows larger, the running time of brute force algorithm grows explosively, while the approximate algorithm shows a good scalability. Figure 7 represents the approximation ratio of the approximate T1U2 algorithm on various value distributions. The approximation ratio is defined as the proportion of the approximated result to the accurate result, i.e. the output of the brute force algorithm. The results show that under either value distribution, the approximation ratio is no smaller than 0.99.

Figure 8 discusses the effect of relaxation ratio β on the running time of both algorithms, when the number of items are fixed at 10. We set ϵ for Algorithm 3 at $1 - \beta$. The running time of



(a) Running time on exponential price distribution (b) Running time on Zipf price distribution

Figure 8: Impact of varying β on running time



(a) Approximation ratio on exponential price distribution (b) Approximation ratio on Zipf price distribution

Figure 9: Impact of varying β on approximate rate

Algorithm 3 increase with β, which well follows the complexity analysis. On the other hand, β does not affect the running time of brute-force method. Figure 9 shows that the actual approximation ratio in practice is much better than the theoretical estimation.

6.3 Top-K Monitoring on Synthetic Dataset

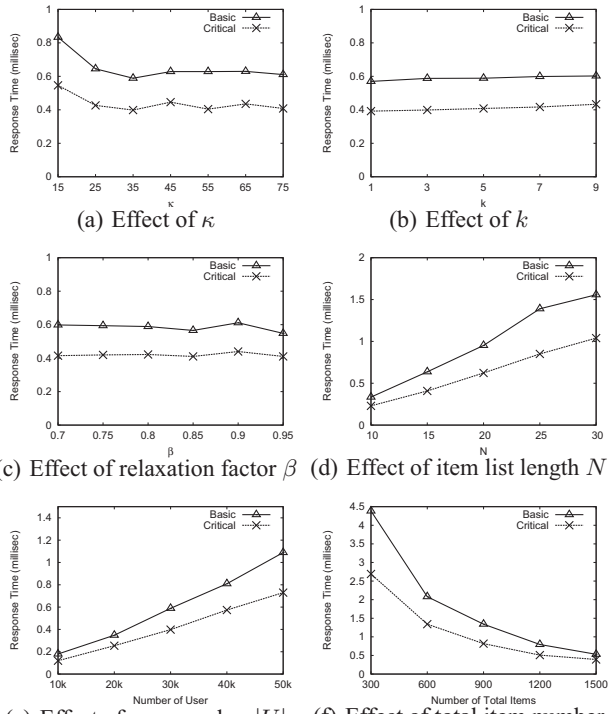
We compare our proposed algorithm with critical item pruning, referred to as 'Critical', with a basic algorithm, referred to as 'Basic'. The basic algorithm is similar to our proposed method. It finds the exchange candidates with the inverted list. However, it does not apply critical item pruning strategy. After exchange candidates are found, the algorithm simply find eligible exchange pairs between current user and each candidate using the T1U2 algorithm.

To verify the efficiency, we measure the response time. Only the experiment results on exponential distribution are summarized, because there is no significant differences among results on various distributions. For each set of experiments, a query file is generated according to the rule we describe in Section 6.1. The query file contains 10 to 30 million updates and is long enough to makes sure that the system finally levels off. The average response time is measured every 1,000 continuous operations. The aim of our experiments is to test the impact of system parameters, the item price distributions and the user number.

As mentioned in Section 5.3, to optimize the performance, the system initially computes the top κ results instead of k, where $\kappa > k$. When one of the old top-k exchanges is deleted, top-κ results are calculated instead of re-computing only top-k results. We first test the impact of the number κ. The empirical result is also used to justify our selection of the default value for κ in Table 5.

The selection of κ affects the system performance on two sides. On the one hand, large κ decreases the frequency of re-computing. On the other hand, it increases the update cost. Figure 10(a) illustrates the system response time when varying κ, when k is set as default value 5. The result shows that the response time reduces when κ increases. The optimal performance is achieved when $\kappa = 35$ for both algorithms. When κ keeps increasing, the system performance levels off, because of the increasing cost of updates.

Then we study the effect of k, i.e. the number of top exchange recommendations. We record the system response time under different values of k. Figure 10(b) shows that the overall response time slightly increases with the growth on k. However, this minor increase makes no significant impact on the overall performance. This implies that the extra overhead brought by increasing k is not an important factor for our system. For basic algorithm, it scans the list and finds the candidate user. Therefore, its running time does not depend on k. For critical algorithm, although increasing k can result in a larger critical item set, the pruning result is not significantly increased. This suggests that our pruning method is effective in reducing the candidate set size.



(a) Effect of κ (b) Effect of k

(c) Effect of relaxation factor β (d) Effect of item list length N

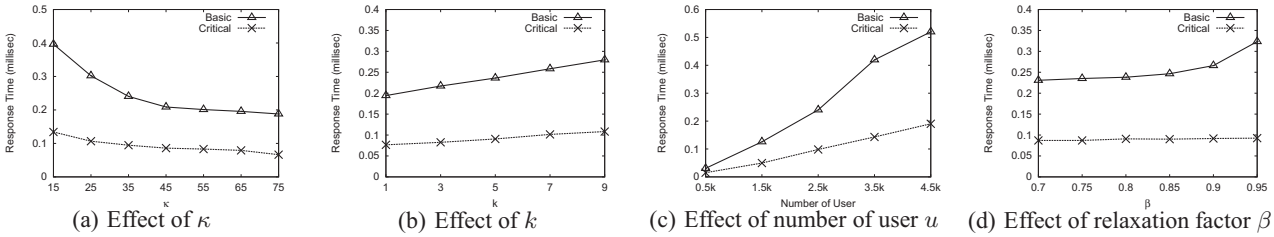
(e) Effect of user number $|U|$ (f) Effect of total item number

Figure 10: Top-K monitoring results on synthetic dataset

We next study the effect of relaxation factor β on the system performance. We illustrate the response time under different β factor, as shown in Figure 10(c). The overall performance always holds on a certain level. This result implies that our system can work well under different β values. Response time of basic algorithm at $\beta = 0.95$ slightly decline in both data sets, since fewer eligible exchange can be found when the relaxation rate is higher.

In our experiments, each user's item list is length fixed. It challenges the system performance when each user is allowed to list more items. We hereby study the performance on different lengthes of item lists. As shown in Figure 10(d), when the item list grows larger, the response time grows linearly with N. When the item list expands, items are more likely to appear in lists for different users. The system has to examine more users to update the exchange recommendations. In practice, users in online communities does not have a long item list. Therefore, the current performance of our system is capable of handling the workload of general community systems.

Number of users in the system is another very important factor which greatly impacts the system performance. We evaluate the response time under different number of users. The result is presented in Figure 10(e). The result shows that the response time linearly grows with the number of users. Despite the decline of the system throughput, the performance of our method is still excellent even for the largest u we have tested (more than 1,000 updates per



(a) Effect of κ (b) Effect of k (c) Effect of number of user u (d) Effect of relaxation factor β

Figure 11: Top-K Monitoring Results on Real Life Dataset

second under 50,000 users).

According to our data generating method, when the number of total items decreases, every item is shared by more users. This brings extra overhead to the system. It is reflected in our test of the system performance with varying number of items. As shown in Figure 10(f), the system performance is inversely proportional to the number of items.

6.4 Top-K Monitoring on Real Dataset

Similarly to the experiments in previous subsection, we compare "Critical" against "Basic" on real dataset. Firstly, we study the effect of κ, which is the initial top results that the system computes. In the tests, k is set at 5. The result is illustrated in Figure 11(a). As can be seen in the figure, response time keeps decreasing with κ increases. For the Basic algorithm, the response time drops significantly before $\kappa = 45$ and levels off after the point. The critical pruning algorithm is not greatly affected by the κ. Its response time decrease insignificantly with κ increases.

Secondly, we study the effect of k, which is the number of top results requested by user. The result is illustrated as Figure 11(b).

The result implies that our pruning strategy can well handle the increasing number of k. For both algorithms, the response time linearly increases with k. The critical algorithm increases slightly slower than the basic algorithm. The overall efficiency shows that our pruning strategy halves the response time. The improvement is better, because in a real life data set, item price distribution is more skewed and user-item ownership are more clustered.

Thirdly, we study the effect of u, which is the number of users participating in the exchange. We test both algorithm under various number of users. As our original (filtered) data set contains 2,458 users, we re-scale the data to generate differently sized data set. We down-scale the data set to generate $u = 500$ and $u = 1,500$ data sets. We up-scale the date to generate $u = 2,500$, $3,500$ and $4,500$ data sets. The result is shown in Figure 11(c).

The result shows that the critical algorithm has a high efficiency and nice scalability. It has an improvement up to near three times. When the user number increases, the response time of critical algorithm grows in a linear manner. Meanwhile, response time of basic algorithm grows faster when user number exceed 2,500. This is because that on the one hand, when we up-scale the data, each item is owned by more user, and the cost of searching for top-k exchange becomes more expensive; on the other hand, each deleting effects more top-k results, which result in a more frequent top-k re-computing. As a result, the basic algorithm shows a super-linear increasing. Since the critical algorithm is less affected by re-computing frequency, it shows a linear growth in response time.

Lastly, we study the effect of β, which is the relaxation factor and also the approximation factor in Algorithm 3. The result is illustrated as Figure 11(b). The critical algorithm perform well under all β, while the response time of the basic algorithm keeps on increasing with β. In a real-life data, user-item ownership are highly clustered. Therefore, small user group often shares a long common item list. In this case, the approximate T1U2 algorithm is launched more frequently than in our synthetic data set. As the approximation algorithm has an time complexity related to $(1-\beta)^{-1}$, the response time increase with β.

7. CONCLUSION

In this paper, we study the problem of top-k exchange pair monitoring on large online community system. We propose a new exchange model, namely Binary Value-based Exchange Model (BVEM), which allows exchange transaction between users only when they both have items the other side wants and the total values of the items are of the same price. We present an efficient mechanism to find the top-1 exchange pair between two users, and extend the analysis to large system with arbitrarily many users. Extensive experiments on synthetic data sets show that our solution provides a scalable and effective solution to the problem.

8. ACKNOWLEDGMENTS

Zhenjie Zhang was partly supported by Singapore A*STAR's Human Sixth Sense Project (HSSP) in Advanced Digital Sciences Center (ADSC).

9. REFERENCES

[1] http://gamersunite.coolchaser.com/games/frontierville.
[2] http://singapore.gumtree.sg/.
[3] http://www.iswap.co.uk/home/home.asp.
[4] http://www.shede.com.
[5] Z. Abbassi and L. V. S. Lakshmanan. On efficient recommendations for online exchange markets. In *ICDE*, pages 712–723, 2009.
[6] D. J. Abraham, A. Blum, and T. Sandholm. Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. In *ACM Conference on Electronic Commerce*, pages 295–304, 2007.
[7] K. Arrow and G. Debreu. Existence of an equilibrium for a competitive economy. *Econometrica*, 22:265–290, 1954.
[8] P. Biró and K. Cechlárová. Inapproximability of the kidney exchange problem. *Inf. Process. Lett.*, 101(5):199–202, 2007.
[9] Y. Chen, S. Chen, Y. Gu, M. Hui, F. Li, C. Liu, L. Liu, B. C. Ooi, X. Yang, D. Zhang, and Y. Zhou. Marcopolo: a community system for sharing and integrating travel information on maps. In *EDBT*, pages 1148–1151, 2009.
[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.
[11] X. Deng, C. H. Papadimitriou, and S. Safra. On the complexity of equilibria. In *STOC*, pages 67–71, 2002.
[12] N. R. Devanur, C. H. Papadimitriou, A. Saberi, and V. V. Vazirani. Market equilibrium via a primal-dual-type algorithm. In *FOCS*, pages 389–395, 2002.
[13] V. V. Vazirani. *Approximate Algorithms*. Springer, 2003.