JAVASLAT A TOP-K ELEMCSERÉK KERESÉSÉRE NAGY ONLINE KÖZÖSSÉGEKBEN

Tanulmány Zhan Su, Anthony K.H. Tung és Zhenjie Zhang: Supporting Top-k elem exchange recommendations in large online communities cikke alapján.

Adatbázisrendszerek elméleti alapjai – ELTE-IK, 2012

Barabás Gábor
Nagy Dávid
Nemes Tamás

ABSZTRAKT

Az egyes elemek kicserélése egyre népszerűbb és széles körben támogatott feladattá válik a nagy online közösségekben például online játékoknál vagy közösségi oldalakon. A hagyományos kézi keresés sem nem hatékony, sem nem hatásos ilyen esetekben. A cserepartnerek automatikus keresésére egyre nagyobb igény van a közösségekben és így potenciálisan új üzleti lehetőségek létrejöttéhez vezet. Az ilyen keresésekhez az usernek meg kell adnia azokat az adatokat amelyekre nincsen szüksége és azokat is,amelyeket keres. Ha az elemek értékét is megadjuk, akkor a cserének két feltétele van: 1. Az egyik usernek pont kell az az elem, amelyik a másiknak már nem és 2. Két hasonló értékű elemet cserélnek. A cikkben új, a keresést támogató struktúrákat mutatnak be különösen azokban az esetekben, amikor az elem listát gyakran frissítik. A javasolt megoldásokat szintetikus és valódi adatokon is tesztelik.

1. BEVEZETÉS

Az egyes elemek kicserélése egyre népszerűbb és széles körben támogatott feladattá válik a nagy online közösségekben, például online játékoknál vagy közösségi oldalakon. Például a *Frontier Ville*-ben, ami az egyik legnépszerűbb online játék több millió játékossal, minden egyes játékos csak meghatározott típusú forrást birtokol. Ahhoz, hogy a játék feladatait meg lehessen oldani, forrásokat kell cserélni [1]. Mivel ennek nincsen egy megfelelő csatornája, most a legtöbb játékos az online fórumokon posztolja a felesleges és vágyott elemeit. De míg ezek csak virtuális tárgyak, számos olyan weboldal van, ahol másodkézből valódi tárgyakat cserélhetünk. Ilyen például a *Schede* Kínában, ami több millió tranzakciót tart számon évente, vagy egyéb oldalak például Nagy-Britanniában [3] vagy Szingapúrban [2]. Ugyanakkor a cserelehetőségeket a userek vagy böngészéssel, vagy keresőszavak beírásával találhatják meg. Annak ellenére, hogy ebben a fajta piacban ekkora lehetőség van, továbbra is óriási szakadék van a felmerülő igények és az ezt támogató technikai lehetőségek között.

Ebben a cikkben megpróbáljuk áthidalni ezt a szakadékot. Általánosságban arról van szó, hogy a lehetséges cserék egy csoportját folyamatosan fenntartjuk és megmutatjuk a felhasználóknak, javasolva a számukra leghasznosabb cseréket. Az online cseréknél a kihívásoknak általában két fajtájával találkozhatunk. Először is egy ésszerű és hatásos cseremodellt kell terveznünk, amit a userek követni akarnak. Másodszor minden ajánlást valós időben kell frissíteni, hogy minden usernek a lehető legjobb cserelehetőséget biztosítsuk, ezt pedig a felhasználóktól jövő tömeges feltöltések kezelése teszi lehetővé.

A felhasználók elvárásainak és viselkedésének modellezésére a közösségi rendszerben, már számos megoldást javasoltak [9]. Egy friss tanulmány [5] például körkörös, egy elemre vonatkozó cserét javasol (CSEM= Circular Single-elem Exchange Model). Ebben a modellben a $\{u_1 \rightarrow u_2 \rightarrow ... \rightarrow u_m \rightarrow u_1\}$felhasználók között létrejöhet egy kör, amelyben minden egyes felhasználó kap a megelőző illetve továbbad a soron következő felhasználónak egy szükséges illetve szükségtelen elemet. A CSEM nagyon hatékony például veseátültetések esetén, de nagy online közösségeknél két okból sem használható. először is a CSEM nem határozza meg az elemek értékét. A csere nem elfogadható akkor, ha az egyik felhasználó nagy értékű elemet ad le, de cserébe csak kis értékűt kap. Másodszor, ha csak egy elemet cserélhetünk, az limitálja a hatékonyságot. Mivel a

CSEM-nek bonyolult a protokollja, a csere csak akkor jöhet létre, ha azzal minden érintett résztvevő egyetért. Ez az online közösségekben nagyon megnyújtaná a várakozási időt. Az 1. ábra mutat egy példát a CSEM hátrányaira. Ebben a példában 3 felhasználó van a rendszerben {u1,u2,u3}, akik vágyott és szükségtelen elemjeit a táblázat soraiban látjuk. A CSEM protokollján alapulva egy valószínű csere egy három useres hurkon keresztül valósul meg, amit az ábra nyilakkal jelöl. Látható, hogy a csere nem éri meg u2-nek, mert I5 100$-t ér, míg I1 csak 10$-t.

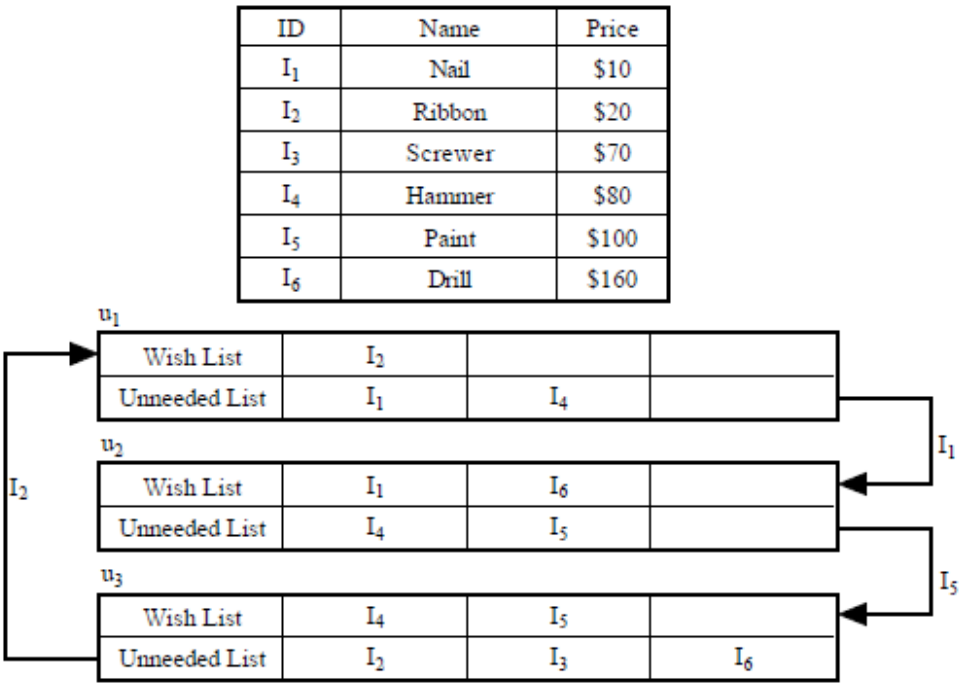


Figure 1: Exampéldáule of transaction in CSEM

Ebben a cikkben egy új modellt ajánlunk, a BVEM-et (Binary Value-based Exchange Model). Ebben a modellben minden csere a rendszer két felhasználója között történik. A csere akkor és csak akkor valósulhat meg, ha a kicserélendő elemek megközelítőleg azonos értékűek. Az előző példát ismét elővéve, ahogy a 2. ábra mutatja, van egy jobb cserelehetőség is u2 és u3 között. Ebben a tranzakcióban u2 180$ értékű elemet ad át és 170$ értékűt kap. A 10$ itt csak 5.9% veszteséget jelent, ami mindkét felhasználónak fair üzlet. Másik részről a BVEM minden cseréje csak 2-2 usert érint, ami nagyon leegyszerűsíti a csere folyamatát. Ez a két tulajdonság nagyon praktikussá teszi a BVEM-et online közösségekben, különösen olyan versenyjellegű felhasználásokban mint az online játékok. A BVEM rugalmasságának és hatékonyságának további fejlesztéséhez egy újfajta lekérdezési elvet javaslunk, a *Top-K Csere Ajánlást (Top-K Exchange Recommendation).* Az elem listák frissítése közben a rendszer megtartja a legjobban értékelt csere elemeket és így ajánlja a legkedvezőbb cseréket.

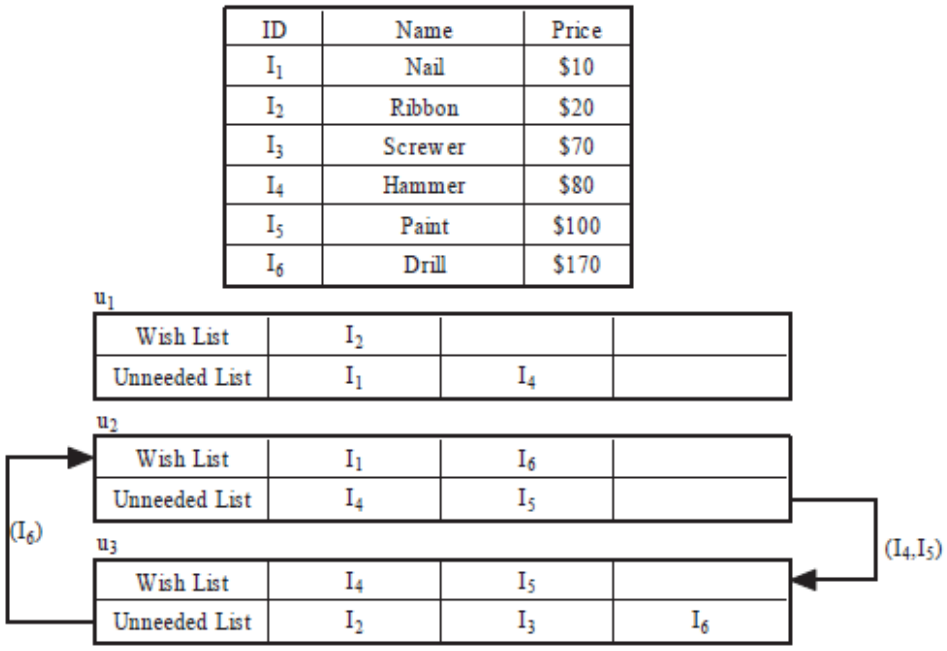


Figure 2: Exampéldáule of transaction in BVEM

A Top-K Csere Ajánlás csábító előnyei ellenére a megfelelő működéshez az adatbázisrendszerek további jelentős fejlesztésekre szorulnak, különösen nagyszámú online felhasználó esetén.

Ha meg van adva két felhasználó a közösségben, a legmagasabb értékkel bíró cserepárok megtalálása NP-nehéznek bizonyul, ahol a számolás kompéldáulexitása a user által birtokolt elemek számával exponenciálisan nő. Szerencsére a legtöbb közösségi rendszerben az elem listák maximális mérete meghatározott, ami elfogadható számolási költségeket eredményez. A probléma bonyolultabbá válik, ha a rendszer dinamikus, azaz a userek gyakran tesznek fel új elemeket illetve törölnek elemeket a listákról. Ahhoz, hogy ezeknek a kihívásoknak eleget tegyünk egy új struktúrát mutatunk be, mely megmutatja a userek-nek a Top-k optimális cserepárokat. A javasolt struktúra a listák frissítésekor is kielégítően működik.

Az cikk tartalmát a következőkben összegezhetjük:

1. Bemutatjuk a Binary Value-based Exchange Model-t, az online cseréknél jellemző viselkedés sajátságainak megragadásával

2. Új adatstruktúrát tervezünk a lehetséges cserepárok hatékony és hatásos indexelésére

3. Optimalizáló technikákat alkalmazunk, hogy növeljük a javasolt struktúra hatékonyságát

4. Széleskörű kísérleti eredményeket vonultatunk fel javaslatunk hasznának ellenőrzésére

A cikk felépítése a következőkben: A 2. szakasz a kapcsolódó munkákat tárgyalja. A 3. szakaszban definiáljuk a problémákat és az ehhez kapcsolódó előfeltevéseket. A 4. szakasz tárgyalja a jelölő struktúrát két user közti cserénél. Az 5. rész ezt több userre terjeszti ki. A 6. rész a javaslatokat szintetikus adathalmazokon teszteli, végül a 7. szakasz vonja le a következtetéseket.

2. KAPCSOLÓDÓ MUNKÁK

Ebben a részben bemutatunk néhány kapcsolódó tanulmányt az informatika különböző területeiről, többek között a vesecsere problémáját az elektronikus kereskedelemben, az online cserélgetős játékok algoritmusát és a csereajánlások problémáját az adatbázis rendszerekben

A vesecsere problémája a vese transzpéldáulantációkból indult ki, ahol sok páciens rokona felajánlotta a veséjét, azonban az nem volt jó az adott páciensnek. Jó ötletnek tűnt cserélni a donorokat a páciensek között. Nagy számú páciens-donor párnál létrejöhetnek olyan körök a párok között, melyek maximális hossza L, és teljesül, hogy minden donor veséje kompatibilis a körben következő páciensnek. Míg a vesecsere problémája NP-nehéz és bonyolult közelítő megoldásokat is találni [8], addig születtek javaslatok egyszerűbb körök keresésére is [6]. Ebben a szerzők egy lineáris egész programozási formulát (ILP) használnak a vesecsere problémájára. Ezzel a járulékos megközelítéssel például helyi optimális megoldásokat találhatunk.

A számítási közgazdaságtanban az Arrow-Debreu Model egy általános reprezentánsa az olyan játékoknak, ahol a résztvevők különböző árucikkeket cserélnek. A játék kezdetén minden egyes résztvevő készpénzt és az árucikkek valamilyen kombinációját birtokolja. Megadva az árucikkek piaci értékét, az userek venni is és eladni is tudnak. Az alap Arrow-Debreu tétel azt állítja, hogy létezik az árak egy olyan kombinációja, amelyik tiszta piachoz vezet, melyben minden user elégedett az elemek végső elhelyezkedésével. Míg a tétel a Kakutani tételével vizsgálja azt, hogy létezik ilyen árkombináció, nem ad arra útmutatást, hogy hogyan találjuk meg ezeket az árakat. A [11, 12]–ben olyan algoritmusokat vizsgálnak, amelyek képesek lehetnek ezen árak meghatározására.

A csereajánlások problémája az adatbázis rendszerekben a vesecsere problémájából származik, és a mi tanulmányunkhoz is közel áll. Abbassi és Lakshmanan az [5]-ben javasolja a vesecserére kifejlesztett CSEM használatát. De ez a CSEM kicsit különbözik az eredetitől, mert itt minden felhasználó különböző árucikkeket is vehet. Továbbá a CSEM-en alapulva további almodellek is kifejleszthetők, mint a Swap Exchange Model, a Short-Cycle Exchange Model és a Probablistic Exchange Model. Az [5] szerzői algoritmusokat ajánlanak, amelyek közelítő megoldásokat találhatnak ezen modellek számára. A mi feltevésünk szerint a CSEM csak akkor praktikus, ha a cseréknek nincs konkrét értéke vagy hatékonysági kritériuma. Mivel az online közösségekben a cserének hatékonyan és gyorsan kell folynia, jobb cseremodellre van szükség.

3. PROBLÉMA DEFINIÁLÁSA ÉS ELŐFELTEVÉSEK

A közösségi rendszerben feltételezzük, hogy van n user: $U = \{u_1,\ldots,u_n\}$ és m elem: $O = \{I_1,\ldots,I_m\}$. Minden u_i usernek két elem listája van: L_i a szükségtelen elemek és W_i a „vágyott" elemek listája. Minden I_j elemhez tartozik egy v_j tag, ami a nyilvános árát jelöli. Legyen $O' \subseteq O$ az elemek egy csoportja. Ekkor O' értéke a benne lévő elemek árának összege: $V(O') = \sum_{I_j \in O'} v_j$. Például az 1-es és 2-es ábrákon a $V(\{I_1, I_2, I_3\})$ = \$100.

Ebben a cikkben a BVEM-et (Binary Value-based Exchange Model) alkalmazzuk a közösségi rendszer cseremodelljeként. u_i és u_l userek $S_i \subseteq L_i$ és $S_l \subseteq L_l$ elemcsoportjai esetén az $E = (u_i, u_l, S_i, S_l)$ cserelehetőség reprezentálja azt az üzletet, amikor u_i minden S_i-ben lévő elemet odaadja u_l-nek és S_l-t kapja cserébe. Az E csere nyeresége u_i számára a kapott elemek összértékével mérhető, azaz $G(E, u_i) = V(S_l)$. Hasonlóan $G(E, u_l) = V(S_i)$ az u_l haszna. A megfelelő csere a BVEM-ben a β ($0 < \beta \leq 1$) relaxációs paraméter használatával mehet végbe, ami az alábbi formális definíciót követi.

Definíció 1: ***Megfelelő cserepár***
Az $E = (u_i, u_l, S_i, S_l)$ cserelehetőség megfelelő, ha kielégíti az 1) összeillő elemek feltételét: $S_i \subseteq W_l$ és $S_l \subseteq W_i$ és a 2) összeillő értékek feltételét: $\beta V(S_i) \leq V(S_l) \leq \beta^{-1} V(S_i)$.

Feltételezve, hogy a rendszer minden usere ésszerű döntéseket hoz, minden u_i user maximalizálni akarja a hasznát a cseréknél. A következőkben bebizonyítjuk, egy egyedülálló cserelehetőség létezését u_i és u_l között, amely mindkettőjük hasznát maximalizálja.

Lemma 1: *Bármely u_i és u_l userpár esetén, létezik egy domináló cserelehetőség $E = (u_i, u_l, S_i, S_l)$ és bármely más $E' = (u_i, u_l, S_i', S_l')$ cserénél a következő két esemény sohasem történhet meg: $G(E', u_i) > G(E, u_i)$ és $G(E', u_l) > G(E, u_l)$.*

Bizonyítás: a lemmát konstrukcióval és indirekt bizonyítással bizonyítjuk. Minden megfelelő cserepárhoz rendelünk (nem emelkedő sorrendben) egy hasznot: $G(E, u_i)$. Azok közül a cserék közül, amik maximális hasznot hoznak u_i-nek kiválasztjuk azt az egy $E = (u_i, u_l, S_i, S_l)$ cserét, amely maximális hasznot hoz u_l-nek is. Ha E nem elégíti ki a lemma feltételét, két eset állhat fenn: 1) van egy E' cserelehetőség, ahol $G(E', u_i) > G(E, u_i)$, de a mi konstrukciónkban ez sosem merülhet fel. 2) u_l-nek jobb lehetősége van, mivel $E' = (u_i, u_l, S_i', S_l')$ csere magasabb hasznot hoz, azaz $G(E', u_l) = V(S_i') > G(E, u_l) = V(S_i)$. Ha ez történik, megmutatjuk a következőkben, hogy $E'' = (u_i, u_l, S_i', S_l)$ szintén egy megfelelő cserepár, így megsérti E konstrukciós elvét. A megfelelő cserepár definícióján alapulva a következőket tudjuk:

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

Mivel $G(u_i, E)$ u_i maximális haszna, bármely cserénél, egyszerű bebizonyítani, hogy $V(S_l) \geq \beta V(S_l') \geq \beta V(S_i')$. Másrészről

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

Ezekből az egyenlőségekből könnyen levezethető, hogy az $E'' = (u_i, u_l, S_i', S_l)$ csere is megfelelő. Ráadásul $G(u_i, E'') = V(S_l) = G(u_i, E)$ és $G(u_l, E'') = V(S_i') = G(u_l, E)$, ami

szintén megsérti a konstrukciós elvet (Lemma 1). Ez az ellentmondás a lemma helyességét támasztja alá. □

A lemma javasolja egy mindkét résztvevő számára optimális cserelehetőség létezését, melyet jelöljön $E^*(u_i, u_l)$. Viszont minden u_i user számára létezhetnek más megfelelő cserelehetőségek egyidejűleg más userekkel is. Hogy további ígéretes cserepárokat kínálhassunk a usereknek, az alábbiakban definiáljuk a *Top-k cserepár* fogalmát.

Definíció 2: ***Top-k cserepár***
u_i usernek a Top-k cserepárok, azaz Top(k, i) tartalmazzák a k legjobb cserepárokat $E^*(u_i, u_l)$ k különböző userrel.

A fentebbi definíció szerint minden userpár legfeljebb egy cserepárral járulhat hozzá Top(k, i)-hez. Ennek oka, hogy van egy domináló csereterv u_i és u_l userek között. Ezért nincs sok értelme két különböző cserejavaslatot tenni egy userpár között. A fő probléma amit meg akarunk oldani az, hogy találjunk egy minden user számára hatékony valós idejű mechanizmust a Top-k cserelehetőségek monitorozására.

Probléma 1: ***A Top-k cserepár monitorozása***
Minden egyes új elem u_i általi L_i-be vagy W_i-be való beszúrásakor vagy törlésekor frissíteni kell Top(k, j)-t, minden u_j user számára.

Minden egyes elem behelyezéskor és törléskor a Top-k párok változás tárgyai lehetnek. A 3-as ábra mutat egy példát ennek a megértésére. A kezdő időpontban csak egy cserelehetőség van u_2 és u_3 között, mégpedig (u_2, u_3, {I_6}, {I_4, I_5}). Ekkor u_3 haszna 180$. A második időpontban – feltéve hogy a csere nem történt meg – u_1 I_3-at hozzáteszi a „kívánság“ listájához. Ekkor u_1 és u_3 között is megfelelő csere jöhet létre. Ennek a cserének a haszna u_3-nak 80$, ami kisebb mint az előzőekben javasolt cseréé u_2-vel. Ennek eredményeképpen az új cserelehetőség u_3 számára a második legjobb javaslat. A harmadik időpontban u_2 kitörli I_5-öt a szükségtelen elemek listájáról. Ez megszünteti az u_2 és u_3 között lévő cserelehetőséget. Fontos megjegyezni, hogy a mi rendszerünk csak javaslatot tesz a usereknek, de sosem viszi végbe a cserét automatikusan.

Az *1-es tételben* bebizonyítjuk, hogy a Top-1 cserepár számolása még akkor is bonyolult, ha csak két user van a rendszerben.

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 Exampéldáule of Top-K Exchange Pair Monitoring with β = 0. 8

Tétel 1: *u_i és u_l userek esetén az optimális cserepár megtalálása NP nehéz.*

Bizonyítás: a problémát visszavezethetjük a már ismert *Load Balancing* problémára. Legyen $X = \{x_1, \dots, x_n\}$ integer-ek egy csoportja. A Load Balancing probléma annak eldöntése, hogy létezik-e X-nek olyan X_1 és X_2 felosztása, hogy $X_1 \subset X$ és $X_2 \subseteq X$ (ahol $X_1 \cap X_2 = \emptyset$ és $X_1 \cup X_2 = X$) akkor $\sum_{x_i \in X_1} x_i = \sum_{x_j \in X_2} x_j$ [13].

A load balancing probléma minden egyes esetében u_i és u_l elem listáját a következők szerint szerkesztjük. $\forall x_j \in X$ esetén a megfelelő I_j elem $v_j = x_j$ értékkel rendelkezik. Minden I_j elem bekerül az u_i W_i listájába és az L_j listába. Az új I_{n+1} elem értéke legyen $v_{n+1} = \sum_{x_j \in X} x_j/2$. I_{n+1}-et beletesszük L_i-be és W_j-be. Ez a redukció O(n) időt vesz igénybe. $\beta = 0$ esetén egy olyan W_i alcsoportot keresünk, melynek az összértéke pont I_{n+1}. Ha egy ilyen megoldást mindig tudunk találni polinomiális idő alatt valamilyen algoritmussal, akkor a load balancing probléma polinomiális idő alatt is megoldható. Ekkor P = NP.

Az utolsó tétel megmutatja, hogy a Top-k cserepárok megtalálásának kompéldáulexitása bármely két user között, exponenciálisan nő az elem lista méretének függvényében. Szerencsére a userek által birtokolt elemek száma a legtöbb online közösségi rendszerben korlátozott. Ez részben egyszerűsíti az optimális cserepárok problémáját. Ezért a legfontosabb probléma a Top-k cserepárok követésénél az, hogy hogyan tudjuk frissíteni a Top-k listáinkat. A cikk további részében néhány olyan adatstruktúrát mutatunk, melyek lehetséges cserepárokat jelölnek és támogatják a gyakori frissítéseket a listákon. Megkönnyítendő az olvasást az 1-es táblázatban definiáljuk a jelöléseket.

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

A következőkben megpróbálunk megválaszolni pár kérdést, ami felmerül az elem csere modellel kapcsolatban.

1-es kérdés: *Lehetséges, hogy a CSEM több cserelehetőséget talál mint a BVEM?* Igaz, de mivel a legtöbb csere értéke nem egyezik, a legtöbb CSEM által megtalált lehetőség nem bír jelentőséggel, például az online játékok szempontjából.

2-es kérdés: *Lehet, hogy az u_i számára javasolt Top-k cserepárok átfedésben vannak?* A BVEM csak javaslatokat tesz a cserére. A valódi rendszer usere a saját preferenciái alapján dönti el, hogy elfogadja-e a cserét vagy nem.

3-as kérdés: *Mi lenne, ha pénzt használnánk mint közvetítő eszközt a userek között?* A valódi vagy a virtuális pénzt nem nagyon használják az online közösségekben. Még ha néhány alkalmazás megengedi is a direkt vásárlást és eladást, a közvetlen csere népszerűbb a userek között, mert így gyorsabban jutnak vágyott elemjeikhez.

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

```
Clear optimal solutions S*
Generate subsets φ_L = 2^{L_i ∩ W_l} and sort on value
Generate subsets φ_R = 2^{L_l ∩ W_i} and sort on value
Set m = |φ_R|
for n from |φ_L| to 1 do
    while m > 0 and β * |φ_R[m]| > |φ_L[n]| do
        m = m − 1
    end while
    if φ_L[n] and φ_R[m] is an eligible exchange then
        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)
    end if
end for
Return S*
```

4. KÉT USER KÖZÖTTI CSERE

Ebben a részben arra a speciális esetre fókuszálunk amikor csak két user van a rendszerben, aki a Top-1 –re érékelt cserelehetőséget keresi. A következő részben kiterjesztjük a vizsgálatot tetszőleges számú userre. A cserét mostantól jelöljük az egyszerűség kedvéért T1U2-vel. Egy ilyen csere megoldható egy offline algoritmussal, melynek a bonyolultsága exponenciálisan nő az elem lista méretének növekedésével.

Az algoritmus a következők szerint működik. Először kiszámolja az átfedéseket a szükségtelen és vágyott elem listák között: $W_i \cap L_l$ és $L_i \cap W_l$. Az algoritmus tesztel minden részhalmaz párt, úgy hogy ezt kielégítse az 1–es definíciót és maximalizálja mindkét user hasznát. A részleteket az 1-es algoritmus tartalmazza. A futás ideje exponenciálisan nő a lista méretével, azaz $O(|S_i|2^{|S_i|} + |S_l|2^{|S_l|})$. Sajnos nem létezik semmilyen pontos algoritmus P kompéldáulexitással, csak ha P=NP. Ezért érdekesebb alternatív megoldásokat keresni, mely közelítő eredményeket ad nagyobb hatásfokkal.

Definíció 3: **ε-közelítő T1U2 csere** u_i számára
Feltételezve, hogy E = (u_i, u_l, S_i, S_l) a legnagyobb értékű cserepár, az E' = (u_i, u_l, S_i', S_l') ε-közelítő u_i számára, ha a haszon nem rosszabb E*-nál 1-ε –nal, azaz G(E', u_i) ≥ (1-ε)G(E*, u_i)*

Az ε-közelítő módszer nem rendelkezik a lemma-1 –ben leírt tulajdonsággal. Ami ε-közelítő cserepár u_i-nek az nem biztos, hogy ε-közelítő lesz u_l –nek is.

A részhalmazok összegzésének problémájánál egy híres polinomiális idejű alkalmazási algoritmus [10] által inspirálva egy teljesen polinomiális idejű közelítési sémát (FPTAS) dolgoztunk ki, hogy kiszámoljuk az ε-közelítő T1U2 cserét. Ráadásul bemutatjuk, hogyan hasznosíthatjuk ezt a megoldást egy újra hasznosítható index struktúra tervezésénél, amely támogatja a frissítéseket.

A közelítő séma ugyan azt az ötletet használja, mint a halmazok összegzése. Általánosságban az eredeti brute-force algoritmus a legtöbb időt az összes lehetséges

elem kombináció generálásával tölti: $W_i \cap L_l$ és $L_i \cap W_l$. Van egy csomó redundáns megoldás, mely majdnem ugyanolyan értékű mint egy másik. Az új algoritmus csak pár elem kombinációt generál $W_i \cap L_j$ és $L_i \cap W_j$ –ben. Ezek a kombinációk megmaradnak a táblázatban a közelítő értékükkel jelölve. A másik elem kombinációk akkor kerülnek a táblába, ha az értékük hasonló a már meglévőkéhez. Az ε-közelítő érték megadásával a V(O‘) pontos értéke egy elem halmaznak γ(O‘) közelítő értékké alakul, garantálva hogy

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

Hogy ezt elérjük ,bevezetjük a következő f(x) kerekítő függvényt. v_{max} és v_{min} a maximális és minimális értékei bármely nem üres elem kombinációnak. Az ε paraméter a hibatolerancia és N jelöli az elemek maximális számát.

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

Ösztönösen f(O‘) az a minimális m egész, ahol $v_{min}\ (1 - \frac{\varepsilon}{N})^{-m} \geq V(O')$. Mivel $v_{min} \leq V(O') \leq v_{max}$ és f(O‘) mindig egész, f(O‘) csak nem negatív egész lehet 0 és $\left\lceil (\log v_{min} - \log v_{max}) / \log(1 - \frac{\varepsilon}{N}) \right\rceil$ között. Ezt a tulajdonságot figyelembe véve impéldáulicit berakjuk az elem kombinációkat N csoportba: $\{S_1, \ldots, S_N\}$. Minden S_m csoport tartalmazza az összes O‘ elem kombinációt, ahol f(O‘) = m, azaz $S_m = \{O' \mid f(O') = m\}$. Minden $O' \in S_m$ elem kombinációhoz tartozik egy γ(O‘) közelítő értékünk, ahol $\gamma(O') = v_{min}(1 - \frac{\varepsilon}{N})^{-m}$. Ez kielégíti az (1) egyenlőtlenséget. Ezek a csoportok fennmaradnak egy reláció táblában, melyet *Közelítő Érték Táblának* (AVT) nevezünk. Az AVT-ben minden AVT[m] bejegyzés statisztikai információt ad S_m csoportnak, hogy megkönnyítse az ε-közelítő T1U2 csere számítását. Specifikusan, arra használjuk az ATV[m].value értéket, hogy jelöljük az általános közelítő értékét minden elem kombinációnak az S_m-ben. Az AVT[m].lb és AVT[m].ub értékeket használjuk minden elem kombinációnak az alsó és felső határának tárolására az S_m-ben. Szintén megtartjuk azokat az elem kombinációkat, amelyek közelítik az alsó és felső határt: AVT[m].lbi és AVT[m].ubi. Erre láthatunk egy példát a 2-es táblázatban.

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

Az AVT tábla létrehozásához minden elemet az ID-jük alapján szortírozzuk. Először az algoritmus inicializálja az első bejegyzést a táblázatba: AVT[0]. Beállítjuk AVT[0].value = AVT[0].lb = AVT[0].ub = 0 –ra és AVT[0].lbi és AVT[0].ubi értékeket üresre. Minden I_j elemre az O’ bemenő elem-halmazban az algoritmus iterálja minden létező AVT[m] bejegyzést az AVT-be és a következők szerint frissíti: minden egyes AVT[m] bejegyzésre az algoritmus megpróbál generálni egy új AVT[n] bejegyzést, ahol n = f(AVT[m].value + v_j). Ha AVT[n] már létezik, akkor az algoritmus megpróbálja I_j-t

betenni AVT[m].lbi-be és AVT[m].ubi –ba, megnézve hogy tud-e generálni új alsó és felső korlátot az S_n csoport számára. Ha AVT[n] nem létezik a táblázatban, akkor az algoritmus új bejegyzést hoz létre. A részletek a 2-es algoritmusban láthatóak.

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

```
1: Generate an empty approximate value table AVT
2: Create a new entry AVT[0]
3: Set AVT[0].lbi = ∅
4: Set AVT[0].ubi = ∅
5: Set AVT[0].value = 0
6: Set AVT[0].lb = AVT[0].ub = 0
7: for each item I_j ∈ O' do
8:    for each entry AVT[m] ∈ AVT do
9:       Calculate M = f(AVT[m].value + v_j)
10:      if there is AVT[n].value = M then
11:         if AVT[m].lb + v_j < AVT[n].lb then
12:            Update AVT[n].lb and AVT[n].lbi
13:         end if
14:         if AVT[m].ub + v_j > AVT[n].ub then
15:            Update AVT[n].ub and AVT[n].ubi
16:         end if
17:      else
18:         Create a new entry AVT[n] in AVT
19:         AVT[n].value = M
20:         AVT[n].lb = AVT[m].lb + v_j
21:         AVT[n].ub = AVT[m].ub + v_j
22:         AVT[n].lbi = AVT[m].lbi ∪ {I_j}
23:         AVT[n].ubi = AVT[m].ubi ∪ {I_j}
24:      end if
25:   end for
26: end for
27: Return AVT
```

Futtassuk az algoritmust egy 3 elemű O‘ = {I_1, I_2, I_3} beállítással, ahol az elemek ára rendre 2, 2 és 3. Ekkor az AVT eredményét a 2-es táblázatban láthatjuk, ahol $(1-\varepsilon/N)^{-1}$ = 2 és v_{min} = 1. Az O‘-ben 7 nem üres kombináció van, melyek {I_1}, {I_2}, {I_3}, {I_1, I_2}, {I_1, I_3}, {I_2, I_3} és {I_1, I_2, I_3}. Az AVT tábla létrehozása után csak 3 bejegyzés van a táblázatban, ami sokkal kisebb mint az elem kombinációk száma. A csoportokra vonatkozó minden információ a táblázat soraiban található. A könnyebb megérthetőség érdekében az utolsó oszlopba betettük a konkrét elem kombinációkat is, bár az AVT nem tartalmazza ezeket.

A következő lemmában megmutatjuk, hogy az AVT kimenete az összes elem kombinációt összegzi ε hibahatáron belül.

Lemma 2: *Megadva bármely O‘ elem halmazra, ahol minden elem kombinációnál O‘‘ ⊆ O‘ a 2-es algoritmus által számolt AVT tábla legalább egy AVT[m] bemenetet tartalmaz, ahol*

$$V(O‘‘) \geq (1-\varepsilon)AVT[m].value$$

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

Bizonyítás: Az egyszerűség kedvéért legyen d $\delta = 1-\varepsilon/N$. Indukcióval belátjuk, hogy $\forall O‘‘ \in O‘$-höz tartozik egy olyan AVT[n], amire igaz a következő:

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

$$AVT[m].lb \leq V(O‘‘) \leq AVT[m].ub \quad (4)$$

Az indukció első lépéseként, ha $|O‘‘| = 0$ vagyis $O‘‘ = \emptyset$, akkor a (3)-as és (4)-es egyenlőtlenségek igazak.

Tegyük fel, hogy a (3)-as és (4)-es állítások az $|O‘‘‘| = k$ –ra igazak. Vizsgáljuk meg, hogy k+1 hosszúságú O‘‘ is áll-e. $O‘‘ := \{ I_1, I_2, \dots , I_{k+1}\}$. A feltételezés által $O‘‘‘ = \{ I_1, I_2, \dots , I_k\}$ –ra van egy AVT[n], amire a (3)-as és (4)-es igaz. A 2-es algoritmus 9-12 sora szerint az AVT tábla I_{k+1} és AVT[n] alapján frissül. Legyen a frissített (11.-16. sor) vagy újonnan alkotott (18.-24. sor) AVT bejegyzés, AVT[m]. Belátható, hogy

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

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

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

Mivel $\delta^k \geq \delta^N = (1-\varepsilon/N)^N \geq 1-\varepsilon$, az állítást bebizonyítottuk. □

Mivel az AVT mérete nem nagyobb mint N, ezért a táblázatot megkonstruáló algoritmus kompéldáulexitása $O(N^2|O‘|)$. Feltételezve, hogy a v_{max}, v_{min}, ε és N ismert konstansok, az algoritmus lineáris időn belül lefut. (Az $|O‘|$ elem mérettől függően.)

Feltételezhetően ez sokkal gyorsabban lefut, mint egy egzakt algoritmus, ha N jóval kisebb mint $2^{|N|}$.

Hasznosítsuk az AVT-t a T1U2 csereproblémánál: két táblázatot, AVT_1-et és AVT_2 hozunk létre az $L_i \cap W_l$ és $W_i \cap L_l$ alapulva. Ha van egy megfelelő cserelehetőség u_i és u_l között, akkor a következő lemma megmutatja, hogy kell hogy létezzen egy $AVT[m] \in AVT_1$ és $AVT[n] \in AVT_2$ pár közeli értékekkel.

Lemma 3: *Ha $E = (u_i, u_l, S_i, S_l)$ egy megfelelő cserelehetőség és $\varepsilon \leq 1-\beta$, akkor léteznie kell két bejegyzésnek: $AVT_1[m] \in AVT_1$ és $AVT_2[n] \in AVT_2$, hogy*

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

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

Bizonyítás: A lemma 2 szerint létezik $AVT_1[m]$ és $AVT_2[n]$ úgy, hogy $AVT_1[m].lb \leq V(S_i) \leq AVT_1[m].ub$ és $AVT_2[n].lb \leq V(S_t) \leq AVT_2[n].ub$. A következő két eset lehetséges:

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

A szimmetria miatt csak az első esetet vizsgáljuk meg. Az egyenlőtlenség bal oldala:

$$\beta AVT_1[m].lb \leq \beta V(S_i) \leq V(S_l) \leq AVT_2[n].ub$$

Az egyenlőtlenség jobb oldala:

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

□

Az utolsó lemma megmutatta, hogy találhatunk cserepárokat a közelítő értéktáblákból azáltal, hogy az alsó és felsőhatárait teszteljük a bejegyzéseknek. A lemmán alapulva mutatjuk be a 3-as algoritmust: hogyan találhatunk ε-közelítő cserepárokat u_i és u_l számára egyszerre. Megadva AVT_1 –et $W_i \cap L_l$ és AVT_2 –őt $L_i \cap W_l$ –en, minden bejegyzés pár $AVT[m] \in AVT_1$ és $AVT[n] \in AVT_2$ tesztelve van. Ha a 3-as lemma feltétele teljesül, akkor két megfelelő cserepárt generál, azaz egyet u_i számára (u_i, u_l, AVT[m].ubi, AVT[n].lbi) és egyet u_l számára (u_i, u_l, AVT[m].lbi, AVT[n].ubi). Az algoritmus ezt követően teszteli ezek optimalitását külön u_i és u_l számára. Miután minden megfelelő cserepárt megtalált, az optimális megoldásokat u_i és u_l számára szeparáltan adja vissza.

Tétel 2: *A 3-as algoritmus lineáris időben adja vissza az ε-közelítő optimális Top-k cserepárokat bármely két user között.*

Bizonyítás: Határozzuk meg a Top-1 cserét (u_i, u_l, S_i, S_l) a lemma 3 által. Találhatunk egy felső (alsó) korlátú elem csoportot: S_i'-t az AVT_1 –ben és egy alsó (felső) korlátú elem csoportot S_l'-t az AVT_2 –ben úgy, hogy ezek megfelelő cserét képeznek és $V(S_i') \geq (1-\varepsilon)V(S_i)$, $V(S_l') \geq (1-\varepsilon)V(S_l)$. Ezért ($u_i$, u_l, S_i', S_l') egy ε-közelítő cserepár. Mivel mind S_i' és S_l' felső vagy alsó korlátú elem csoportok és a 3-as algoritmus minden felső/alsó korlát értékpárt összehasonlít, az S_i' -t és S_l' -t biztosan megtaláljuk. □

A 3-as algoritmus írja le a közelítő T1U2 keresését. Mivel bármelyik táblának maximum N bejegyzése van, az algoritmus kompéldáulexitása $O(N^2)$. Ha minden bejegyzést a közelítő érték csökkenő sorrendjében és a szkennelő bejegyzéseket fentről-lefelé rendezzük, az időigényt O(N)-re csökkenthetjük.

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

5. ÁLTALÁNOS TOP-K CSERE

Ebben a részben a közelítő értéktáblát használjuk, hogy megtaláljuk a Top-1 cserelehetőséget. A valódi rendszerekben 2-nél jóval több user van, így kiterjesztjük a tárgyalásunkat meghatározott számú userre. A probléma egyenes megoldása az |U|(|U|-1) közelítő értéktáblázatok karbantartása. Minden egyes u_i, u_l user párra két közelítő értéktábla AVT_{il} és AVT_{li} jön létre és frissül folyamatosan $W_i \cap L_l$ és $L_i \cap W_l$ szerint. Bármely frissítésnél az u_i listáin a rendszer újraszámítja a T1U2-t u_i és bármely másik u_l között. Top(k, i) és Top(k, l) úgyszintén az új optimális csere szerint frissül. Sajnos ez a módszer nem használható nagy közösségi rendszerek esetén, mert a táblázatok száma négyzetesen nő. A felhasznált memória csökkentése végett nem dinamikusan frissítjük a táblákat a user párok között. Helyette beépítünk pár könnyűsúlyú indexet, melynek helyigénye lineáris az indexek számával. u_i listáinak frissítésével ez a rendszer megtalálja minden egyes u_l-t, aki esetleg érintett Top(k, i) vagy Top(k, l) cserében. Hogy ezt elérjük, először pár szükséges feltételt adunk meg a Top-k cserepárokra vonatkozólag a *Critical Elem Set* koncepciójával.

Definíció 4: *Adott u_i user W_i elem listája esetén az elemek $O' \subseteq W_i$ alcsoportja egy kritikus alcsoportot alkot, ha $V(W_i) - V(O') < G(u_i, Top(k, i))$.*

Más szavakkal az O' elem alcsoport kritikus W_i szempontjából, ha a többi W_i-ben lévő elem összértéke nem nagyobb mint u_i jelenlegi optimális haszna. A következőkben K_i –t használjuk a kritikus elem csoportok jelölésére. Lássuk be, hogy a 4-es definíció csak egy elégséges feltételt ad a kritikus elem csoportra. W_i elem lista esetén kombinációk százai elégíthetik ki a fenti definíciót. Az 5.1.-es részben megmutatjuk, hogyan lehet létrehozni egy jó kritikus elem csoportot néhány kritérium felhasználásával.

Lemma 4: *Ha Top(k, i) tartalmaz egy E = (u_i, u_l, S_i, S_l) cserepárt, akkor S_i legalább egy I_j elemet tartalmaz a K_i kritikus elem csoportból W_i –re vonatkozóan.*

Bizonyítás: Feltételezzük, hogy S_i nem tartalmaz elemet a K_i –ben: $S_i \subset W_i - K_i$. Ezért $V(S_i) \leq V(W_i) - V(K_i) < G(u_i, Top(k, i))$. Ez ellentmond annak a feltételnek, hogy S_i egy Top-k csere. Ezért S_i legalább egy elemet tartalmaz bármely kritikus elem csoportban. □

A lemma 4 azt jelenti, hogy a rendszernek csak akkor kell újra számolnia a T1U2 cserét u_i és u_l között a Top(k, i) –t frissítéséhez, ha u_l legalább egy kritikus elemét tartalmazza u_i –nek és fordítva. Ez indokolja a mi index struktúránkat, amely a kritikus elemek fordított listáján alapul. Két fordított lista van minden egyes elemen: $CL(I_j)$ és $UL(I_j)$. $CL(I_j)$ tartalmazza azokat a usereket, akiknél I_j a kritikus elemek között van és $UL(I_j)$ tartalmazza azokat a usereket, akiknél I_j a szükségtelen elemek között van. Általánosságban, ha van egy frissítés a W_i listán, akkor a rendszer megkap egy lehetséges user csoportot a fordított listákról és kiszámolja a T1U2 cserét. Ennek részletes leírása a 4-es algoritmusban van. A lemma 4 szerint ez az algoritmus nem téveszt egyetlen frissítést sem a Top listákon.

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

```
1: Clear the left candidate user set CU_l
2: for each I_j in the critical item set of W_i do
3:     merge UL(I_j) into CU_l
4: end for
5: Clear the right candidate user set CU_r
6: for each I_j ∈ L_i do
7:     merge CL(I_j) into CU_r
8: end for
9: for each u_l ∈ CU_l ∩ CU_r do
10:    Compute T1U2 between u_i and u_l
11:    Update Top(k, i) and Top(k, l) accordingly
12: end for
```

5.1. Kritikus Elem Kiválasztása

Az optimális kritikus elemet a 4-es algoritmus szerint választjuk ki. Ha meg van adva W_i, rengeteg módon alkothatjuk meg a K_i kritikus elem csoportot. Általánosságban egy jó kritikus elem csoport leredukálja a 4-es algoritmusban tesztelt userek számát. Hogy ezt elérjük, pár költségmodellt mutatunk be.

$UL(I_j)$ tartalmazza azokat a usereket, akiknél I_j a szükségtelen elemek között van. Alapvetően feltételezzük, hogy $|UL(I_j)| << |U|$. Továbbá feltételezzük, hogy $UL(I_j)$ különböző elemek esetén nem korrelál erősen, azaz $|UL(I_j) \cap UL(I_k)| << |UL(I_j)|$. Eszerint az ellenőrizendő userek száma $\sum_{I_j \epsilon K_i} |UL(I_j)|$ szerint becsülhető. A fenti analízisen alapulva a jó kritikus elem csoport a következőkkel adható meg:

$$\sum_{I_j \epsilon K_i} |UL(I_j)| \longrightarrow MIN$$

$$\sum_{I_j \epsilon K_i} v_j \geq V(W_i) - G(u_i, Top(k,i))$$

Bár ez a probléma NP teljes, adható egy szinte optimális megoldás egy egyszerű *mohó* algoritmussal: a W_i elemjeit csökkenő sorrendbe rendezzük, majd egyesével szelektáljuk az elemeket míg összértékük meg nem haladja a $V(W_i) – G(u_i, Top(k, i))$ értéket.

A 3-as táblázatban erre láthatunk egy példát 5 userrel.

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: Exampéldáule of critical elems sets of 5 users

5.2. Elem Behelyezése a Listába

Elem behelyezés esetén a rendszer megkapja az összes lehetségesen érintett usert és újraszámolja a T1U2 cseréket, hogy frissíteni tudja a Top-k ajánlásokat

Miután I_j bekerült a W_i listába, pár új cserelehetőség generálódik. Ha van új megfelelő cserelehetőség u_i és u_l között, akkor I_j L_i –n is rajta van. Ezért a lehetséges CU user csoport inicializálódik az $UL(I_j)$ listával. Minden CU-ban lévő u_l részére a rendszer megvizsgálja, hogy u_i –nek van-e u_l részére kritikus eleme vagy fordítva. Ha ezek bármelyike fennáll, akkor a 3-as algoritmus keresi meg az optimális cserét kettejük között.

A 3-as táblázatban erre láthatunk egy példát az u_2 W_i listájába az I_1 elem betevésével. Ekkor u_3 –at és u_5 –öt kellene megvizsgálnunk, mivel $I_1 \in L_3, L_5$.

5.3. Elem Törlése a Listáról

Ha I_j –t W_i –ről eltávolítjuk, a művelet két lépésben valósulhat meg. Az első lépésben a rendszer kitöröl minden olyan Top-k cserét, mely ezt az elemet tartalmazta. A második lépésben a rendszer új cserelehetőségeket keres a userek számára.

Az első lépés fordított listákkal valósul meg, ami lehetővé teszi a rendszernek, hogy gyorsan megtalálja az összes érintett k cserepárt. Feltételezzük, hogy a userek a törölt cserepárokkal mind egy rögzített userlistában vannak. Ekkor a 4-es algoritmus hívódik meg, mely a listában lévő user számára fixálja a Top-k javasolt párokat. Feltételezhetjük, hogy a törlési funkció drága, ha sok usert adunk a a javítandó userek listájához.

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

```
1: Clear the left candidate user set CU_l
2: for each I_j in the critical item set of W_i do
3:     merge UL(I_j) into CU_l
4: end for
5: Clear the right candidate user set CU_r
6: for each I_j ∈ L_i do
7:     merge CL(I_j) into CU_r
8: end for
9: for each u_l ∈ CU_l ∩ CU_r do
10:    Compute T1U2 between u_i and u_l
11:    Update Top(k, i) and Top(k, l) accordingly
12: end for
```

Ahhoz, hogy optimalizáljuk a rendszer működését, különböző technikákat javaslunk, melyek redukálják a userek számát a fixált userlistában a törlés után. Ennek alapötlete, hogy megtartjuk a Top-κ cserepárokat minden u_i user számára a következő egész számmal: $\kappa > k$. Bizonyítható, hogy Top(k, i) részhalmaza a Top(κ, i) –nek. Hogy hasznosítsuk a kiterjesztett Top-κ csereajánlást, a rendszer frissíti a Top(κ, i) –t minden egyes új elem behelyezésekor. Elem törléskor, ha a cserepárok egyike $E \in$ Top(κ, l) eltávolítódik $I_j \in W_i$ törlése által, a cserelista nem frissül azonnal. Ehelyett a rendszer az új T1U2 csere értékét becsüli meg. Ha az új optimális csere a Top(κ, l) –ben marad, akkor ez direkt visszahelyeződik a Top(κ, l) –be. Egyébként a számláló eggyel csökken κ-ról κ-1 –re. A Top(κ, l) teljes újraszámítását elhalasztjuk, amíg u_l nem frissíti valamelyik listáját, vagy k-nál kevesebb cserepár marad a rendszerben. Bebizonyíthatjuk, hogy minden cserepárt a Top(k, i) –ben a rendszer folyamatosan megtart. Bár a behelyezéseknél több költség merül fel, a jelenlegi optimalizáció nagyban javítja a rendszer működését, mert kiküszöböli a felesleges újraszámítását a Top cserepároknak.

6. KÍSÉRLETEK

Ebben a részben értékeljük az előzőekben bemutatott algoritmust. Szintetikus és életszerű adatokon is teszteljük a rendszert, utóbbiakat a B2B online piacról vesszük.

6.1. Adatgenerálás és Kísérleti Beállítások

6.1.1. Szintetikus Adathalmaz

A szintetikus adatgenerálás első lépése bizonyos számú elem megalkotása. Minden elemhez rendelünk egy értéket. Az értékeket meghatározott eloszlások szerint generáljuk, mint exponenciális és Zipf eloszlások. Az összes eloszlás paramétereit a 4. táblázat tartalmazza. A maximum értéket 10000-re, a minimumot 10-re állítjuk. Az értékek meghatározásánál az algoritmus az árakat ezen két érték között tartja.

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

A valódi rendszerekben a userek és elemjeik általában szorosan korrelálnak, a hasonló ízlés és viselkedés miatt. Hogy a diverzitást és a csoportképző tulajdonságot megragadjuk, 5 osztályt hozunk létre és ezzel modellezzük a különböző usertípusokat és népszerű elemjeiket. Minden usert random módon, azonos valószínűséggel rendeljük hozzá az egyes csoportokhoz. Az osztályok egyikét háttér-osztálynak nevezzük, ebben minden elem benne van. Minden elem a másik négy osztály valamelyikébe is tartozik egyenlő valószínűséggel. Az elemek számának maximalizálására minden listának van egy N felső határa. A lista akkor van tele, ha az elemek száma eléri ezt a felső határt. Hogy a rendszer skálázhatóságát teszteljük, kísérletünkben igyekszünk a listákat tele tartani.

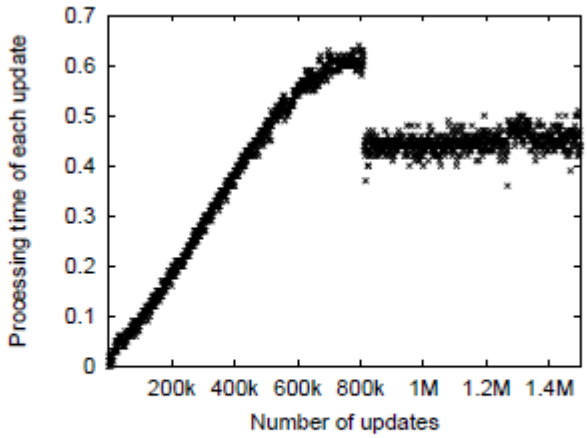


Figure 4: Average update response time over time

Miután beállítottuk a paramétereket és a usereket és elemeket is osztályokba soroltuk, sorozatos elemfrissítésekkel szintetikus adatokat generálunk. Ez két fázisból áll. Az első a bemelegítő fázis. Ennek célkitűzése, hogy minden user vágyott és szükségtelen elem listáját feltöltsük, tehát itt több hozzáadás van, mint törlés. Ezt követően a második fázisban elkezdődik a szimuláció, melyben a törlések és hozzáadások frekvenciája hozzávetőlegesen ugyanez, ami relatíve stabil munkaterhelést eredményez a rendszeren.

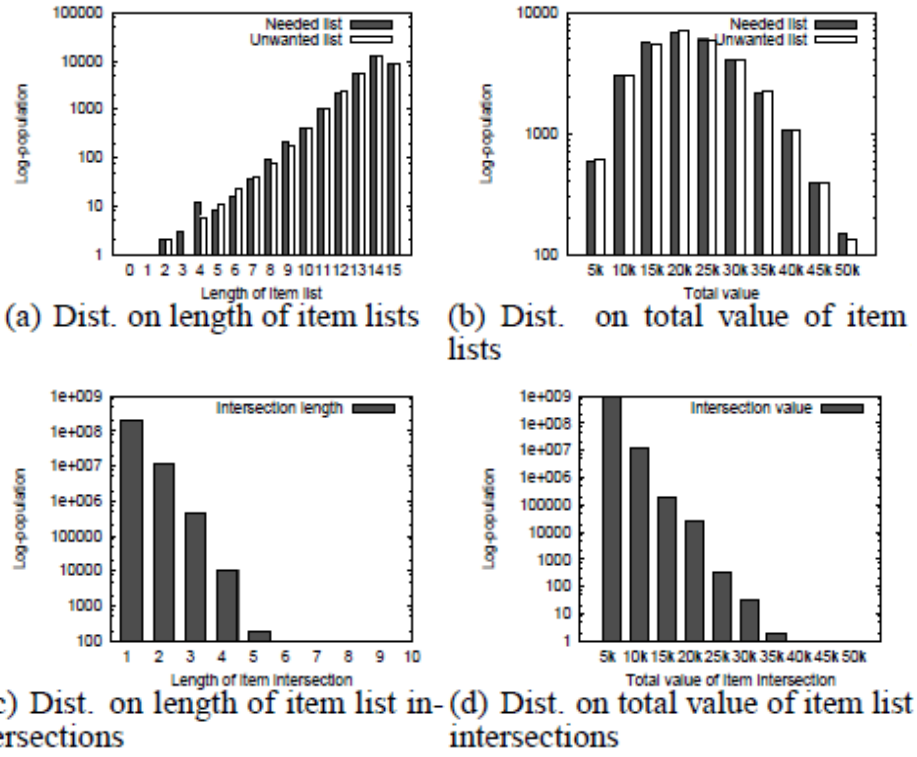
Az első fázisban, mikor új frissítést generálunk, a szimuláció random választ a userek közül. A generátor ezután választ egy vágyott vagy egy szükségtelen listát. Ha a céllista nincs tele, akkor betesz egy elemet. Egyéb esetben random módon töröl egy elemet. A behelyezendő elem kiválasztása a user és az elem osztályától is függ. A generátor random módon választ egy számot, hogy eldöntse, hogy az elem és a user azonos osztályból (4/7 valószínűség), a háttérosztályból (2/7 valószínűség), vagy másik három osztályból (1/7 valószínűség az összes és 1/21 minden egyes osztályra) való-e. Aztán uniformizáltan

választ egy elemet az adott osztályból. A törlés során azonos valószínűséggel választ egy elemet a listáról, ez az adott elem osztályától független.

A második fázisban, az elsőhöz hasonlóan a választott user egy elem listályát választjuk ki. Ha üres, akkor beteszünk egy elemet. Ha sem nem üres, sem nem tele, a generátor randomizált döntést hoz: 0,6 valószínűséggel betesz, 0,4 valószínűséggel kivesz egy elemet. Ezek a valószínűségek biztosítják a listák telítettségét.

Az első fázisban generált frissítések száma N*|U|, ahol |U| a userek száma és N az elemek maximális száma bármely listán. A második fázisban generált frissítések száma sosem kevesebb, mint 2*N*|U|. A teljesítmény pár frissítés után stabillá válik a második fázisban.

Az 5.ábrán az átlagos frissítés válaszidejének fejlődését mutatjuk be a szimuláció során. Az első fázisban a válaszidő gyorsan emelkedik. Amikor a második fázisba érünk, a teljesítmény stabillá válik. Minden eredményt a második fázisból kapunk.



(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 elem lists and intersections

Az 5. ábra illusztrálja az elemek megoszlását bizonyos futási idő és a rendszer stabilizálódása után. A rendszerben 30000 user van és az elem lista hossza maximum 15 elem lehet. Az 5(a) ábra szemlélteti az elem lista hosszúságának userek közti megoszlását. Látható, hogy a userek nagy részének majdnem teljes– több mint 80%-nak 13,14 vagy 15 elemű - listája van. Az 5(b) ábra mutatja az egyes userek elem listáinak érték szerinti megoszlását. Amint látható, a teljes érték 15k és 20k körül csoportosul. Az 5(c) az elem listák átfedéseinek hosszát mutatja,ahol az átfedés a két user mindegyike által birtokolt elemeket jelenti. Látható, hogy az átfedés általában kicsi, a legtöbb esetben nem több, mint 5 elem. Hasonló trend látható z 5(d) ábrán, ami az átfedő értékek userek

közötti megoszlását rajzolja ki. Minden $|U|^2$ userpár között csak pár száz olyan userpár van, ahol az átfedés meghaladja a 20k értéket.

Az 5. táblázat összegzi a kísérleten tesztelt paramétereket. A default értékek vastagon vannak szedve.

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. Valódi Adathalmaz

Nehéz valódi csereadatokat szerezni az online közösségekből. Mi az eBay.com-ról szereztünk tranzakciós adatokat, ami az egyik legnépszerűbb C2C online piac. 90 napig folyamatosan gyűjtöttünk adatokat meghatározott userektől. Összességében 34191 usert, 452774 elemet és 1094152 tranzakciót regisztráltunk. A kívánt és szükségtelen listákat a vásárolt és eladott elemekből állítottuk össze.

Mivel az online piac különbözik a cserepiactól, előre feldolgoztuk az adatokat, hogy a rendszerünk számára használhatóvá tegyük őket. Azt találtuk, hogy sok elem dupéldáulán szerepel és nagyon sok a nagyon hasonló elem. Ezt redukálandó, a nagyon hasonló elemeket összevontuk. Pár elemtől és usertől megváltunk, hogy ne legyen üres elemlista. Mindezek után 2458 user és 2769 elem maradt.

Ahhoz, hogy a rendszerünk működését tesztelhessük változó számú user között, újraskálázzuk az adatokat, hogy változó méretű adatcsomagokat kapjunk. Ehhez random módon dupéldáulázzuk a userek számát a kívánt méret eléréséig. Ezekhez a userekhez hozzárendeljük az eredeti userekhez tartozó elemeket. Az eredeti és a dupéldáulikált adatokból véletlenszerűen választunk usereket.

Folyamatos frissítéseket generálunk, a már létrejött tranzakcióknak megfelelően: véletlenszerűen választunk egy usert, egy tranzakciót és egy elemlistát (vágyott/ szükségtelen). Egy elemlista maximális hosszúsága 15 elem.

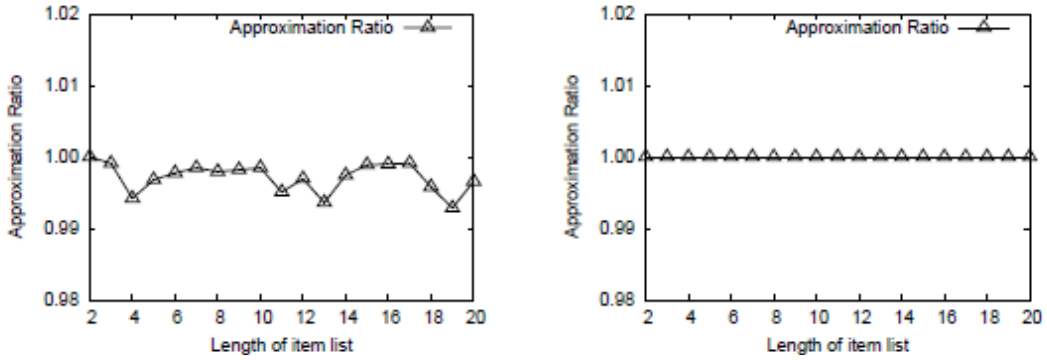
A 6-os táblázat mutatja a tesztelt paramétereket. A default értékek kövéren vannak szedve.

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. Kísérletek T1U2 Cserével

A 4. részben bemutattuk a 3-as algoritmust, amely egy közelítő eljárást adott a T2U1 csere megtalálásához. Ebben a részben megvizsgáljuk és értékeljük a működését, beleértve a futási időt és a közelítés pontosságát. Ehhez egy brute-force módszert fogunk használni. Mindkét algoritmust teszteljük exponenciális és Zipf eloszláson.



(a) Approximation ratio on exponential price distribution

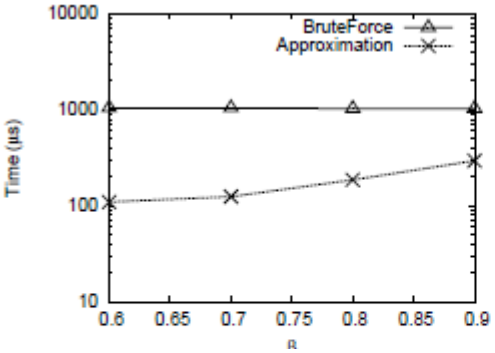
(b) Approximation ratio on Zipf price distribution

Figure 7: Impact of varying elem list length on approximation

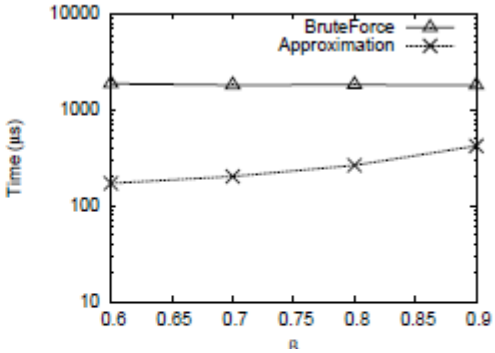
A 6. és 7.ábra mutatja mindkét algoritmus működését különböző hosszúságú elemlisták mellett. Rögzítjük β-t és (1-ε) –t is 0.8 –on, továbbá két egyforma hosszú elemlistát generálunk: $W_i \cap L_l$ és $L_i \cap W_l$. A 6-os ábrán a két algoritmus futásidejét láthatjuk: ha az elemlista mérete kevesebb mint 8 elem, a közelítő séma nem olyan jó mint a brute-force. Ennek oka, hogy a köztelítés túl sok időt tölt az indexet megalkotásával. Ugyanakkor nagyobb számú elemnél a brute-force algoritmus futási ideje exponenciálisan nő, míg a közelítő algoritmus jól skálázható. A 7-es ábrán a közelítő T1U2 algoritmus *közelítési arányát* láthatjuk különböző eloszlások mellett. A *közelítési arány* definíció szerint a közelítő érték és a valódi érték hányadosa. A valódi értéket a brute-force szolgáltatja. Az eredmények azt mutatják, hogy ez az arány sosem kisebb mint 0.99.

A 8-as ábra a β relaxációs érték mindkét algoritmus futási idejére gyakorolt hatását mutatja. Az elemlisták elemszáma 10. ε-t a 3-as algoritmus számára (1-β) –nak állítjuk be. Az algoritmus futási ideje a β-val együtt növekedik, ami jól követi a feladat kompéldáulexitását. A brute-force algoritmusnál a β nem befolyásolja a futásidőt. A 9-es

ábra megmutatja, hogy az aktuális közelítési arány a gyakorlatban sokkal jobb, mint az elméleti becslés.

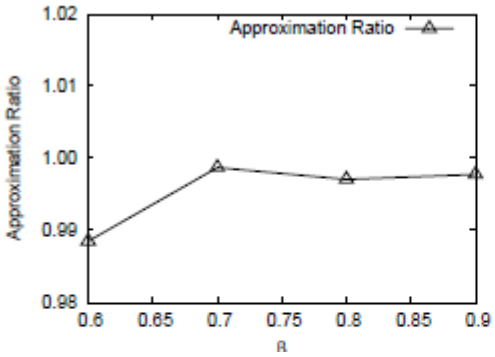


(a) Running time on exponential price distribution

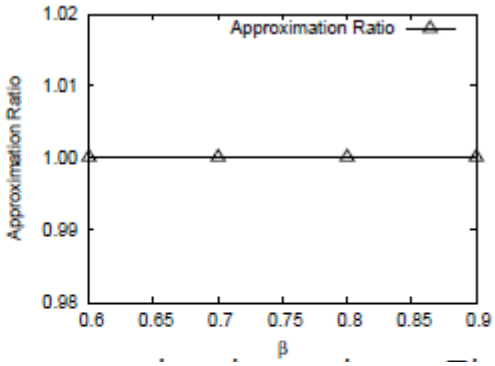


(b) Running time on Zipf price distribution

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

6.3. Top-K Monitorozás Szintetikus Adathalmazzal

Összehasonlítjuk az előzőekben bemutatott algoritmust – továbbiakban *kritikus* algoritmus – egy alap algoritmussal. Az utóbbi hasonló a javasolt metódushoz. Fordított listák segítségével találja meg a userek lehetőségeit, de nem használ „kritikus elem" stratégiát. Miután megtalálta a cserelehetőségeket, az algoritmus egyszerűen a T1U2 algoritmus segítségével találja meg a cserepárokat.

Hogy megvizsgáljuk a hatékonyságot, mérjük a válaszidőt. Csak az exponenciális eloszlás eredményeit összegezzük, mert nincs számottevő különbség az egyes eloszlások között. Minden kísérleti beállításra egy lekérdező fájl generálódik a 6.1. részben leírt szabályok szerint. Ez a fájl 10-30 millió frissítést tartalmaz. A kísérlet célja, hogy tesztelje a rendszer paramétereinek, az elemek árának és a userek számának a hatását.

Mint az 5.3 –ban említettük, a működés optimalizálásához a rendszer a Top- κ értékeket számolja k helyett, ahol $\kappa > k$. Ha valamely régi Top-k elem törlődik, akkor a rendszer a Top-κ –t számolja a Top-k elemek újraszámolása helyett. Ezért először a κ értékét növeljük. Az empirikus eredmény egyben az 5-ös táblázatban bemutatott default κ értékek helyességének ellenőrzésére is szolgál.

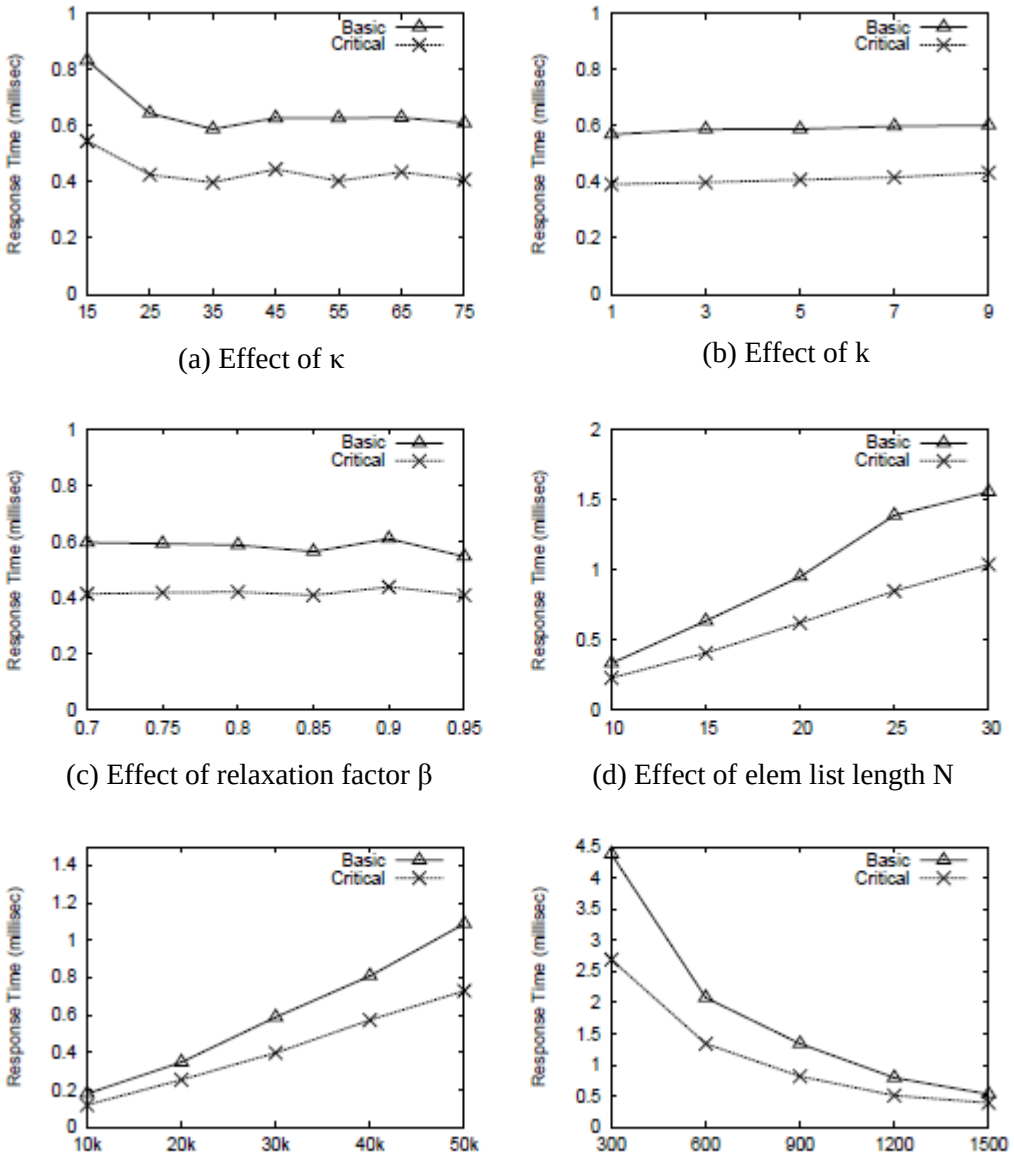


Figure 10: Top-K monitoring results on synthetic dataset

κ megválasztása két oldalról is befolyásolja a rendszer működését. Egyrészt ha nagyra állítjuk, azzal csökkentjük az újraszámolás gyakoriságát. Másrészt ez emeli a frissítés költségét. A 10(a) ábra szemlélteti a rendszer válaszidejét a κ változtatásával k = 5 default érték esetén. Az eredmény azt mutatja, hogy a válaszidő a κ növelésével csökken.

Mindkét algoritmus működése κ = 35 esetén optimális. Ha κ –t tovább növeljük, a rendszer működése romlik, mivel a frissítések költsége túl magasra nő.

Most vizsgáljuk meg k –nak, a top csere javaslatok számának a hatását. Különböző értékek mellet mérjük a rendszer válaszidejét. A 10(b) ábra mutatja, hogy k növelésével a válaszidő nő, de ez a kis növekedés a rendszer szempontjából nem számottevő. Az alap algoritmus scan-neli a listát és megtalálja a lehetséges usert. Ennélfogva futási ideje k-tól nem függ. A kritikus algoritmusnál bár a k emelése eredményezhet nagyobb elem-alcsoportokat, a „purning“ eredmény nem emelkedik szignifikánsan.

A következőkben vizsgáljuk a β relaxációs faktort. A 10(c) ábrán a válaszidőt láthatjuk különböző β –k mellett. Az általános működés mindig azonos idejű. Tehát a rendszer jól működik különböző β –k mellett. Az algoritmus válaszideje β = 0.95 –nél enyhén csökken mindkét adathalmaznál, mert kevesebb megfelelő usert lehet találni, ha a relaxációs ráta magasabb.

Kísérleteinkben minden user elemlistája fix hosszúságú. Ezek bővítése kihívás elé állítja a rendszert. Mint a 10(d) ábra mutatja, ha az elemlista nagyobbra nő, a válaszidő N-nel lineárisan fog növekedni. A növekedéssel egyre valószínűbb, hogy egy-egy elem több user listáján is megjelenik, ezért a rendszernek a lehetséges userek megtalálásához több usert kell átvizsgálnia. A gyakorlatban az online közösség usereinek elemlistája általában nem hosszú, így a rendszerünk ekkor jól használható.

A userek száma ismét egy fontos tényező a rendszer működése szempontjából. A 10(e) ábra mutatja a működést a userek számának függvényében. A növekedéssel az átlagos válaszidő lineárisan nő. Azonban még a legmagasabb értékeknél is kiváló eredményeket mutatott a rendszer: 10000 frissítés/sec és 50000 user.

Az adatgenerálási módszerünk szerint, ha az összes elem száma csökken, minden elemet több user oszt meg. Ez extra overhead –et visz a rendszerbe. Ezt mutatja az elemek számának változtatásának a rendszer működésére gyakorolt hatása. Ahogy a 10(f) ábra mutatja, a rendszer működése fordítottan arányos az elemek számával.

6.4. Top-K Monitorozás Valódi Adathalmazzal

Mint az előző részben, újra összehasonlítjuk a kritikus és alap modellt egymással, azonban most valódi adatok segítségével. Először is a κ értékét vizsgáljuk. A tesztekben a k = 5 beállítást alkalmazzuk. Az eredményeket a 11(a) ábra mutatja. Az alap algoritmus válaszideje szignifikánsan csökkent κ = 45 –ig, ám ezt követően ez abbamaradt. A kritikus algoritmusra nem volt számottevő hatással a κ érték növelése.

Ezt követően a k értéket vizsgáljuk, melyet a 11(b) ábrán láthatunk. Az eredmények azt mutatják, hogy a k növelésével mindkét algoritmus válaszideje lineárisan növekedett. A kritikus algoritmus növekedése lassabb mint az alap módszeré és folyamatosan az előbbi értékei alatt marad. A szintetikus adatok esetén rosszabb eredményeket kaptunk

Harmadszorra az u értékének változását vizsgáljuk, mely a cserékben résztvevő userek számát adja meg. Mindkét algoritmust teszteltük. Az eredeti modell 2458 usert tartalmazott. Az alapos teszteléshez különböző további adatcsomagokat generáltunk kevesebb (u = 500, u = 1500) és több userrel egyaránt (u = 2500, 3500, 4500). Az eredményeket a 11(c) ábra mutatja.

Az eredmények azt mutatják, hogy a kritikus algoritmus hatékonyabb és skálázhatóbb mint a másik. A növekedés lineáris. Az alap algoritmusnál a válaszidő gyorsabban növekszik u = 2500 –nál, majd lelassul u = 3500 –nál.

Utoljára a β relaxációs faktort vizsgáljuk meg, mely a 3-as algoritmusnál szerepelt. Az eredményeket a 11(d)-n láthatjuk. A kritikus algoritmus minden β esetén jobb válaszidőt produkál mint az alap.

7. KONKLÚZIÓ

A fentiekben a Top-k cserepárok monitorozásának problémáját tanulmányoztuk nagy online közösségi rendszerek esetén. Bemutattunk egy új cseremodellt, a BVEM-et. Ez a modell csak akkor javasolja a usereknek a cserét, ha mindkettejüknek van olyan eleme, amelyre a másiknak szüksége van és ezek értéke közel megegyező. Bemutattunk egy hatékony mechanizmust a két user közötti Top-1 cserepár megkeresésére és kiterjesztettük az analízist meghatározatlan számú usert tartalmazó nagy rendszerekre is. Szintetikus adatokon végeztünk alapos kísérleteket és ezekkel bizonyítottuk, hogy megoldásunk skálázható és hatékony megoldást kínál a keresett problémára.

8. REFERENCIÁK

[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.