Tipikus időbeli internetezői profilok nagyméretű webes naplóállományok alapján

Schrádi Tamás
schraditamas@aut.bme.hu
Automatizálási és Alkalmazott Informatikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

Kivonat: A cikk a webes naplóállományokból történő időbeli internezői profilok kialakításának műszaki problémáit ismerteti, két lehetséges feldolgozási módszert vizsgál futási idő, memóriahasználat és a feldolgozás során hasznosítható egyéb tulajdonságok (pl. hibatűrés) szempontjából, valamint a kialakított profilokból meghatározza a tipikusnak mondható felhasználói csoportokat. A böngésző sütijein alapuló technika segítésével előálló, nagyméretű adathalmaz alapján határozzuk meg az időbeli felhasználói profilokat, majd ezeken hajtunk végre klaszterezést. A felhasználói profilok kialakítása az ismertetett módszerekkel akár egy személyi számítógépen is megtehető, hibatűrő módon, majd a klaszterek kialakításával az adathalmazban meglévő információt alakítjuk emberi felhasználás számára átláthatóbbá, értelmezhetőbbé.

Kulcsszavak: out-of-core feldolgozás, klaszterezés, cookie alapú felhasználó azonosítás, időbeli profil, tipikus internetezők

1. Bevezetés

A mindennapi munkamenet során a Web szerverek a felhasználói kérésékről alapvető adatokat rögzítenek, azonban ezeket a lehető legalacsonyabb absztrakciós szinten, hiszen elsődleges feladatuk a felhasználói kérések kiszolgálása. Az elemi szinten tárolt adatokat a későbbi elemzés során mindig a megfelelő alakúra kell alakítanunk, mielőtt az elemzés megkezdhetnénk.

A webes naplóállományok nagy mérete a feldolgozó algoritmusok futási idő szerinti optimalizálását még hangsúlyosabbá teszi, míg a megnövekedett futási idő megköveteli a feldolgozó algoritmus hibatűrő tulajdonságát. A továbbiakban nagy mennyiségűnek számítanak azok az adatok, amik egy időben nem férnek el az elsődleges memóriában.

Annak érdekében, hogy a feldolgozást egy átlagos memória mennyiséggel rendelkező számítógépen is végre lehessen hajtani, a cikk két másodlagos memóriát is használó algoritmust is ajánl. A módszerek ismertetésén és futási idejének elméleti vizsgálatán túl, az algoritmusok futási idő szerinti optimalizálását is tárgyalja, figyelembe véve a memória korlátossága okozta megkötéseket. A szakaszos feldolgozás egy hibatűrő feldolgozó módszer, ami a háttértárolóra menti a feldolgozott naplóbejegyzéseket összefésülve a meglévő eredményekkel. A k-utas összefésülésen alapuló technika első feldolgozási lépésként minden állományt külön feldolgoz, majd a keletkező részekre k-utas összefésülés segítségével határozza meg az időbeli profilokat. Az időbeli profilok nagy száma miatt emberek számára még nem használhatóak jól a kapott profilok. Az egyedi profilokon már alkalmazható egy adatbányászati feldolgozás, aminek segítségével meghatározhatóak a tipikus felhasználói csoportok. A kialakított felhasználói profilok alapján klaszterezés segítségével határozhatjuk meg a tipikus felhasználói csoportokat. Az így kialakult profilok hasznosíthatóak szolgáltatói oldalon (pl. a különböző csoportokhoz leginkább illő reklám megjelenítésével).

A felhasználók azonosítása alapvető fontosságú minden további feldolgozás előtt. Ez a böngészők sütijeit (cookie) használó technikán alapszik, az ezekben a sütikben kiosztott azonosítók alapján kerülnek naplózásra a felhasználói interakciók. A naplózás egy központi

szerveren történik, ami több auditált Web helyet is összefoghat. A felhasználói azonosítás az alábbiak szerint történik: a Web helyek közvetett, third party cookie-kat (C3) raknak le a kliensek böngészőjébe (1x1pixel méretű háttérszínű kép, ami egy hivatkozás a vizsgálatot végző szerverre). Ez a hivatkozás az oldal első letöltésekor egy, a központi szerver által kiadott cookie-t tesz le a kliens böngészőjében. Ez a C3 azonosító egyedi és azonos lesz az összes naplózott webhely meglátogatása során. Ha egy tetszőlegesen figyelt oldalra látogat a felhasználó, akkor a böngésző felküldi a C3 sütijét a központi szerverre, így megoldható a felhasználó követése.[9][10] Azonban egyre gyakoribb, hogy a felhasználók törlik a third-party cookie-kat, ezzel jelentősen torzítva a mérési adatokat, hiszen ebben az esetben a felhasználó számára egy új C3 sütit kell kiosztanunk, holott ugyanaz a felhasználó.

Azért, hogy ezt a problémát elkerüljük, kiegészíthetjük a naplózást az ún. közvetlen, first party cookie-k használatával (C1 cookie). Ebben az esetben az a hivatkozás helyett egy kódrészletet helyez el a naplózásért felelős web hely a vizsgált oldalon és ez a kódrészlet generálja a web hely nevében a C1 cookie-t. Jellemző, hogy a C1 sütik sokkal ritkábban kerülnek törlésre, ezért ha ezt az azonosítót is elküldi a böngésző a szerverhez fordulások esetén, akkor az ugyanazon a felhasználóhoz tartozó, de újragenerált C3 cookie-k a C1 cookie-k mentén összekapcsolhatók, ezzel téve lehetővé a pontosabb felhasználói profilok kialakítását.

Felmerülhet a kérdés, hogy kit tekintünk felhasználónak az elemzés során. Mivel a böngészőkben a kétfajta cookie felhasználónként külön kerül tárolásra, ezért a felhasználót az alábbi hármas írja le: az operációs rendszer, az operációs rendszerbe belépett felhasználó azonosítója és maga a használt böngésző. E kifinomultabb internet felhasználói definíció segítségével sokkal pontosabb profilokat, statisztikákat és kimutatásokat lehet készíteni, mint az egyszerű IP cím alapú felhasználó azonosítással.

A szétszórtan meglévő kattintások alapján kerül kialakításra a felhasználói profil. Ez egy olyan összetett adatstruktúra, amelyben a felhasználó kattintásainak számát, aktivitási időtartamát, kattintásainak naponkénti, illetve óránkénti megoszlását tároljuk. Az elemi naplóbejegyzés tartalmazza a C3 azonosító mellett az időbélyeget is, így ez alapján előállítható, illetve módosítható a felhasználói profil.

A cikkben vázolt kutatási téma mögött egy valós probléma áll: webes naplóállományok feldolgozása. A kapcsolódó projektben rendelkezésünkre állnak valós, magyarországi Web helyekhez tartozó naplóállományok, amelyek alapján a tipikus felhasználói csoportokat meg kell határoznunk. Csupán egyetlen hónaphoz tartozó naplóállomány több mint 160 GB-nyi elemi adatot jelent a háttértáron, ez több mint 6 milliárd 32 és 64 bites számok formájában megjelenő bejegyzést jelent (több mint 1500 állományban eltárolva). Ilyen formájában biztosan nem használható fel az adathalmaz, ezért az emberi felhasználás számára értelmezhetővé kell tennünk.

A cikk az alábbi szakaszokból épül fel: az első szakasz ismerteti a megoldandó problémát, a második szakaszban a felmerülő problémák esetén alkalmazható módszerek, ezek lehetséges optimalizációja kerül bemutatásra, a harmadik szakasz foglalkozik a tipikus felhasználói csoportok kialakításával, a negyedik szakaszban az eredményeket tekinthetjük meg, míg az utolsó szakasz összefoglalja a munkát és lehetséges továbbfejlesztési irányokat ismertet.

2. Az előfeldolgozás problémái

A naplózással előállított adathalmaz rengeteg, elemi bejegyzést tartalmaz, amit ilyen formájában biztosan nem tudunk feldolgozni. Ezért első lépésben át kell alakítanunk az adathalmaz, az egy felhasználóhoz tartozó kattintásokat egy ún. időbeli felhasználói profilba kell aggregálnunk. Az előfeldolgozás során a kihívást nem a feldolgozó algoritmus bonyolultsága okozza, hanem az a tény, hogy az adathalmaz mérete meghaladja a rendelkezésre álló operatív tár méretét, ezért az a feltételezés, hogy a feldolgozás során

elférünk az elsődleges memóriában, biztosan nem fog teljesülni. Annak érdekében, hogy az elsődleges memória szabta korlát ellenére is feldolgozhassuk az adathalmazunkat ún. out-of-core módszereket alkalmazhatunk a feldolgozás során.

Az adatbányászati módszerek gyakran során előkerülnek az out-of-core módszerek, ahol ún. partícionálással elv alapján oldják meg a feladatot számos cikk szerint [6][7][8]. Eszerint az elv szerint a nagy adathalmazt feldarabolják kisebb, a memóriában egy időben elférő darabokra, ezekre elvégzik a műveleteket, majd a másodlagos memóriában tárolják a részeredményt. Az eltárolt részeredmények alapján származtatják a teljes adathalmaz feldolgozásával előálló transzformált adatokat. Az out-of-core módszerünk akkor alkalmazható, ha a feldolgozás során alkalmazható a particionálási elv. A felhasználói profilok kialakítása esetén ez az elv alkalmazható: a kisebb méretű adathalmazok feldolgozásával előálló részeredményeket összefésülve, frissíthetjük a kattintásokat a megfelelő eloszlásban. Out-of-core módszerek az adatbányászati területek mellett, vizualizációs problémák esetén is jól alkalmazható (számítógépes grafika, tudományos adatmegjelenítés) [11][12].

Az out-of-core módszerek jellemzője, hogy maguk a módszerek fel vannak rá készítve, hogy az adatok nem fognak elférni az operatív memóriában, ezért másodlagos memóriát (tipikusan merevlemez) is felhasználnak a működésük folyamán. Mivel nagy adathalmazok kerülnek feldolgozásra, ezért a hatékonyság alapvető fontosságú. Azonban egy fontos, a jelenlegi merevlemez és operatív memória elérése közötti különbséget is figyelembe vegyünk, mert ugyanazt az adatot sokkal gyorsabb érhetjük el az operatív tárban, mint a merevlemezen. Ha a futási időre akarunk optimalizálni, akkor érdemes a merevlemez műveleteket alacsonyan tartani.

Annak érdekében, hogy az out-of-core módszerek létjogosultságát alátámasszuk, egy, csak az operatív tárat használó algoritmussal is megpróbáltuk feldolgozni az adathalmazt. Az algoritmus az elsődleges memóriában tárol egy jól méretezett hash alapú struktúrát, majd a naplóállomány elemeit végigolvasva létrehoz, illetve módosít felhasználói profilokat. Ez a megközelítés az adathalmaz csak egy részét képes hatékonyan feldolgozni, ezzel nem is a konkrét feldolgozás volt a célunk, hanem egy viszonyítási alapot akartunk képezni.

A következő két szakasz röviden ismertet egy-egy ilyen módszert, optimalizálási lehetőségekkel.

2.1. Szakaszos feldolgozás

A szakaszos feldolgozás során az adathalmazt osszuk fel azonos méretű blokkokra, amik már elférnek az elsődleges memóriában. A feldolgozási lépések során egy-egy ilyen blokk kerül feldolgozásra az alábbi elvek szerint. Az első lépésben feldolgozzuk az első blokkot, majd mentjük a feldolgozott adathalmazt a merevlemezre. Mivel elemek milliói között kell hatékonyan keresnünk, ezért egy hash alapú tárolási struktúrát használunk a memóriabeli feldolgozás folyamán. Egy jól méretezett hash O(1) idejű keresést tesz lehetővé a tárolt elemek között. A feldolgozás további lépéseiben az aktuális blokkot is feldolgozzuk, azonban a mentés során az új részeredmény a meglévő részeredmény és az aktuálisan feldolgozott eredmény összefésülésével áll elő. Így folyamatosan propagálja az algoritmus a feldolgozott adatokat a merevlemezen, majd az utolsó lépés után megkapjuk a teljes adathalmaz feldolgozásával előálló profilokat. Hogy az összefésülést hatékonyan megoldhassuk valamilyen rendezettség szükséges az elemek között (pl. az azonosítók szerint növekvő sorrend szerinti rendezés). Feltételezve, hogy az adathalmazunk n rekordot tartalmaz, amit m méretű blokkokra osztunk, az algoritmus háttértár műveleteinek a száma az alábbi képlettel fejezhető ki: $c_{\text{háttértár}} = n + \frac{\bar{\alpha}}{m} n^2$, ahol $\bar{\alpha}$ a feldolgozás alatti átlagos tömörítési tényezőt jelenti. A feldolgozási komplexitás az alábbiak szerint számítható ki:

$c_{paros} \approx \left(\frac{f(m)}{m} + \frac{\bar{\alpha}\cdot\beta}{2}\right)n + \frac{\bar{\alpha}\cdot\beta}{2m}n^2$, ahol a β a szorzótényező, ami az összehasonlítás és az elvégzendő műveletet összemérhetővé teszi, $f(m)$ pedig az m méretű blokk feldolgozásának ideje.

Az algoritmusban meg kell választanunk a blokkméretet, azaz az egyszerre feldolgozandó adatmennyiséget. Minél nagyobbra választjuk a blokkméretet, annál kevesebb I/O műveletre lesz szükség a feldolgozás során, annál gyorsabb lesz a feldolgozás. Azonban fontos, hogy a program által felhasznált memóriát kézben tartsuk, megelőzve a kilapozás okozta teljesítménybeli csökkenést. Így az egyes blokkok méretnövelésének határt szab az az igény, hogy elférjünk az elsődleges memóriában. Az egy időben feldolgozandó blokkok számát megadhatjuk úgy is, hogy korlátozzuk az algoritmus által felhasználható memória mennyiségét. Az algoritmus ezen változatát adaptív szakaszos feldolgozásnak nevezzük. A gyakorlat azt támasztja alá, hogy az adaptív szakaszos feldolgozás segítségével hatékonyan dolgozhatunk fel nagy adathalmazokat, anélkül, hogy erre a feladatra egy szuperszámítógépet használnánk.

A nagy adatméret hosszú futási időket eredményez, jellemző, hogy a futási idő, órákban mérhető, nem másodpercekben. A megnövekedett futási idő azt is jelenti, hogy a feldolgozástól független hibák következhetnek be, amik a feldolgozási folyamatot megszakíthatják. Ha csak a belső memóriát használó algoritmusunkban még nem mentettük a feldolgozott adatokat, akkor elveszítjük a kialakított profilokat, kezdhetjük elölről a feldolgozás. A szakaszos feldolgozás egy hasznos tulajdonsága, hogy a háttértárra történő mentések miatt automatikus hibavédelmet is tartalmaz az algoritmus: ha valamilyen váratlan hiba folytán megszakad a feldolgozás, nem kell elölről kezdenünk a feldolgozást, hiszen csak az aktuálisan feldolgozás alatt álló profilok vesznek. A már kimentett profiloktól kezdve folytathatjuk a feldolgozást.

2.2. K-utas összefésülésen alapuló feldolgozás

A k-utas összefésülés alapú módszer ugyancsak azonos méretű blokkokra osztja fel az adathalmazt, majd feldolgozza ezeket a blokkokat. Az algoritmus a minden egyes blokk után menti a háttértárra valamilyen reláció szerint rendezetten az adatokat, azonban a feldolgozás után mást nem tesz. Ha minden blokk feldolgozásra került, akkor a részeredményekből a végleges eredményt egy k-utas összefésülés segítségével állítja elő.

A módszer háttértárműveleteinek költsége az alábbiak szerint alakul:
$c_{disk} \approx 2\bar{\alpha}\cdot n\cdot \log_k n + \left(1+\bar{\alpha}(1-2\log_k m)\right)n$, ahol k az összefésülésnél használ utak száma. A futási idő komplexitása $c_{proc} \approx \bar{\alpha}\cdot n\cdot \log_k n + \left(\frac{f(m)}{m} - \bar{\alpha}\cdot\log_k m\right)n$.

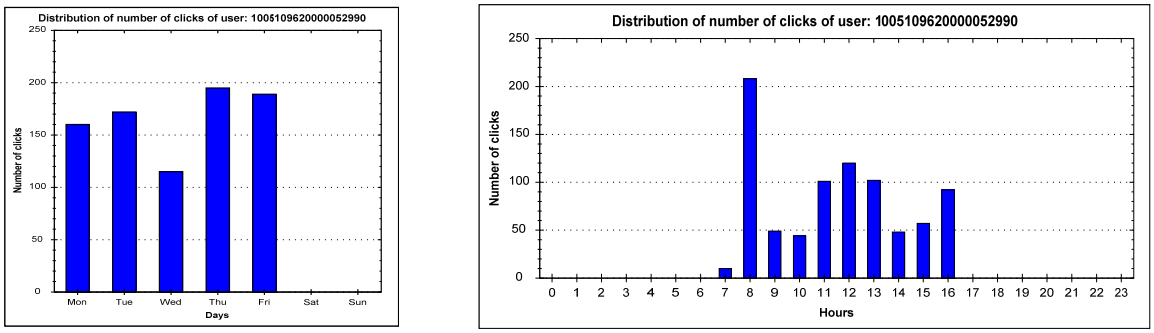
E módszer esetén is érdemes a blokkméretet nagyra választani, úgy, hogy még elférjenek az adatok az operatív tárban. Egy további optimalizációs lehetőséget jelent a k-utas összefésülés során az utak számának növelése. Elvileg minél több utas összefésülést alkalmazunk, annál hatékonyabb lesz a feldolgozás, de itt is figyelembe kell vennünk egy gyakorlati szempontot, mégpedig azt, hogy a sok utas összefésülés folyamatos pozicionálást jelent a merevlemezen, ami a teljesítménycsökkenéshez vezet.

Hasonlóan a szakaszos feldolgozáshoz, ez a módszer is beépített hibatűréssel rendelkezik, egy külső hiba folytán nem a teljes feldolgozott adathalmaz fog elveszni, hanem csak az adathalmaz éppen a memóriában lévő része. A feldolgozás ezek után folytatható a meglévő háttértáron található feldolgozott adatok alapján.

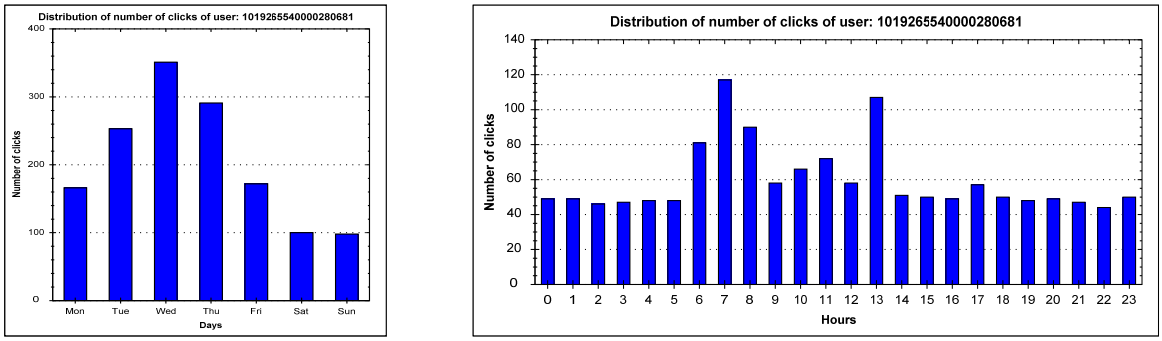
A két módszert összevetve, elmondhatjuk, hogy a k-utas összefésülésen alapuló módszer háttértárigénye nagyobb, hiszen az egyes blokkok a feldolgozás után rögtön a háttértárra mentődnek.

3. Tipikus felhasználói csoportok kialakítása – klaszterezés

Az előfeldolgozási lépés során az elemi adatokból aggregálással kialakítottuk az időbeli felhasználó profilokat, ezzel az emberek számára sokkal értelmezhetőbb adatokat származtattunk a meglévő adatokból. Tekintsük az 1. ábra és a 2. ábra, ami két felhasználói profilt szemléltet: az 1. ábra egy tipikus munkahelyen internező felhasználó időbeli profilját ismerhetjük fel, hiszen a naponkénti kattintás-megoszlási grafikonon látható, hogy a hétvégén nincs felhasználói aktivitás, az órákra bontott kattintás megoszlásban pedig a kattintások a 7-16 óra közötti idősávba esnek. A 2. ábra egy másik típusú felhasználói profilt jelenít meg: e mögött valamilyen automatizmust tételezhetünk fel, mert az óránkénti eloszlásban megfigyelhető minta nem lehet hosszú távú, emberi viselkedés.



1. ábra: Egy munkahelyen internetező időbeli profilja



2. ábra: Egy automatizált viselkedést sejtető profil

Annak ellenére, hogy a feldolgozási lépéssel sikerült az adathalmazunkat csökkentenünk, a keletkező profilok száma túl nagy ahhoz, hogy emberek számára jól átlátható legyen, információt hordozzon (több mint 12 millió profil készült egy havi naplóállomány feldolgozása során). Az adataink absztrakciós szintje már értelmezhetőség szempontjából a célnak megfelelő, azonban a nagy számosság még mindig alkalmatlanná teszi az emberi értelmezésre. Így egy megoldás lehet, hogy ha meghatározzuk a tipikus felhasználói csoportokat, majd ezeket a csoportokat jellemezzük, hiszen az üzemeltetőknek sokkal értékesebb információt jelentenek a tipikus profilok, mint az egyéni profilok.

A feladatot adatbányászati feladatra lefordítva, a meglévő adathalmazunkban kell csoportokat kialakítanunk, úgy, hogy az egymáshoz hasonló profilok egy csoportba kerüljenek. Az adatbányászati feladatok közül pontosan ezt célozza meg a klaszterezés, ami egy struktúrafeltáró többváltozós elemzési eljárás, melynek segítségével objektumok tulajdonságaikban hasonló csoportjait alakítjuk ki. Klaszterezés (clustering) alatt az elemek csoportosítását értjük, ahol előre nem adottak a csoportok és úgy szeretnénk kialakítani a csoportjainkat, hogy az azonos csoportba eső elemek jobban hasonlítsanak egymásra, mint a más klaszterbe kerülőkhöz. A klaszterezés az adatbányászat egyik legrégebbi feladata, ezért már igen „régen” születtek algoritmusok a klaszterezés feladatának megoldására. Ha a gépi tanulás szemszögéből közelítjük meg a klaszterezést, akkor ez felügyelet nélküli tanulásnak

tekinthető (unsupervised learning). Az osztályozás feladatától elsősorban ez különbözteti meg: a klaszterezés során nincsenek tanítópéldák, amikre adott lenne a klaszterezés, nincs közvetlen visszacsatolás a klaszterezés jóságáról.

Jon Kleinberg 2002-ben publikált cikkében bebizonyítja, hogy olyan távolság alapú klaszterezés nem létezik, ami kielégít három, általa definiált tulajdonságot.
A három tulajdonság, amelyek egy klaszterezés jóságát hivatottak megragadni a következők [5]:

1. Skála-invariancia: ha minden elempár távolsága helyett annak α-szorosát (α>0) vesszük, akkor a klaszterezés eredménye ne változzon.
2. Gazdagság: tetszőlegesen megadott csoportosításokhoz tudjunk megadni távolságokat, hogy a klaszterező algoritmus az adott módon csoportosítson.
3. Konzisztencia: amennyiben az algoritmusunk előállítja a pontok egy csoportosítását, amelyet felhasználva a klaszteren belül a távolságokat lecsökkentjük, míg a különböző klaszterbe került elemek közötti távolságot megnöveljük, akkor az újonnan kapott távolságok alapján az algoritmus az eredetivel megegyező csoportokat állítson elő.

A Kleinberg-féle lehetetlenség elmélet tétele: ha a klaszterezendő elemek száma nagyobb, mint egy, akkor nincs olyan távolság alapú eljárás, ami rendelkezik a skála-invariancia, gazdagság és konzisztencia tulajdonságaival [5]. Azonban, ha a gazdagság és a konzisztencia tulajdonságán is lazítunk, akkor belátható, hogy létezik olyan algoritmus, ami teljesíti a skála-invariancia, a lazított gazdagság és a finomítás-konzisztencia tulajdonságot[5].

A klaszterező algoritmusokból igen sokat találunk az adatbányászati szakirodalomban, ezek közül az egyik leggyakrabban alkalmazott és aktívan kutatott algoritmus a k-közép. A k-közép (k-means) az egyik legrégibb klaszterező eljárás, ami vektortérben adott adatokat klaszterez [1][2]. Az algoritmus a fizikai rendszerek súlypont analógiájára épít. A klaszterezés jóságának megragadására a négyzetes hibafüggvényt használja; a klaszteren belüli szórást, azaz a négyzetes hibafüggvényt próbálja minimalizálni. A négyzetes hibafüggvény $\sum_{i=1}^{k}\sum_{x_j \in S_i}(x_j - \mu_i)^2$,ahol S_i, μ_i az i-edik klasztert, illetve az i-edik klaszter középpontját (centroidját) jelöli, k klasztert kell kialakítani és $x_j \in S_i$ jelöli az i-edik klaszterbe eső elemeket.

Az algoritmus első lépésben kiválasztja a kezdeti klaszterközéppontokat. Az eredeti k-közép algoritmusban ez véletlenszerűen történik. Ezután minden elemet besorol a hozzá legközelebb eső klaszterközépponttal jellemezett klaszterbe. Ez egy első besorolása az elemeknek. Következő lépésben az egy klaszterbe került elemek átlagát képezve újabb klaszterközéppontokat állít elő az algoritmus, amelyeknek segítségével újból besorolja az elemeket a klaszterekbe. Az új klaszterközéppont kiválasztását és az elemek besorolását iteratívan ismétli, mindaddig, amíg egy leállási feltétel nem teljesül. Ilyen leállási feltétel lehet például, hogy az elemek besorolása nem változik, vagy az elemek besorolása mindössze ε-nyit változik. Az algoritmus során az olyan helyzetekben, amikor azonos távolságra van egy pont a klaszterközéppontoktól valamilyen rögzített, determinisztikus elv alapján kell döntetnünk (pl. előfordulás szerint az első klaszterközépponthoz). Ez azért fontos, hogy az ilyen határhelyzetbe került pontok besorolása ne változzon, ne állhasson fenn a folyamatos átsorolás lehetősége.

Az algoritmus elliptikus alakú klasztereket próbál meg kialakítani és minden egyes pontot egyértelműen egy klaszterhez rendel. Nagy előnye, hogy egyszerű és a gyakorlatban jól skálázható, az algoritmus futási ideje $O(n \cdot k \cdot t)$, ahol n a klaszterezendő pontok száma, k a kialakítandó klaszterek száma, míg t az iterációk száma [1][2]. A gyakorlatban tapasztalt jól

skálázhatósága teszi alkalmassá nagy adathalmazok kezelésére is. Annak ellenére, hogy az algoritmus a gyakorlatban gyorsan konvergált, mutattak olyan adathalmazokat, amelyekre az elemek számában a lineáris komplexitásnál nagyobb komplexitás látható be [3].

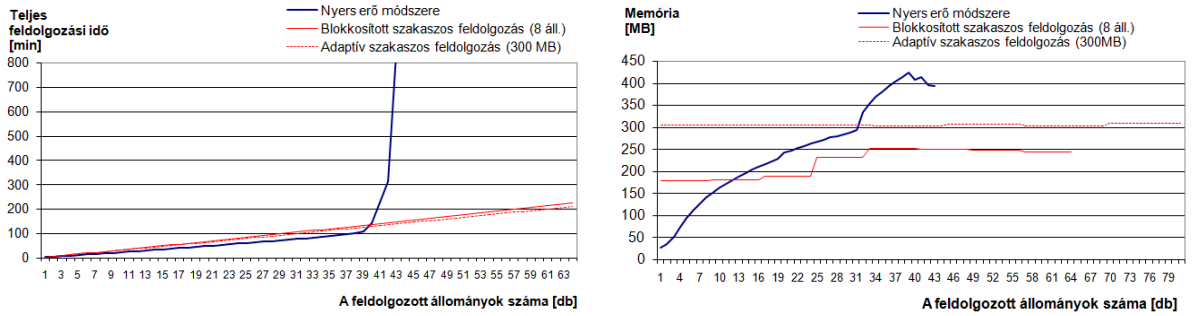
Mivel a k-közép algoritmus a kezdőpontokat véletlenszerűen választja ki, ezért lokális optimumba ragadhat a klaszterezés. A lokális optimumba ragadást az algoritmus többszöri lefutásával lehet kezelni és a legkisebb négyzetes hibát adó klaszterezést tekintjük a legjobbnak.

A kialakított profilokat úgy értelmezzünk, mint többdimenziós (37 dimenziós), numerikus adatok, majd ezeken az értékeken hajtjuk végre a k-közép algoritmust. Az esetlegesen nem numerikus típusú adatokat ahhoz, hogy alkalmazhassuk rajtuk a k-közép algoritmust numerikussá kellene transzformálnunk, de ez a jelen esetben nem áll fenn.

Mint minden valós adathalmaz, az adathalmazunk is zajjal terhelt ezért a zajt is kezelnünk kell a klaszterezés során. Mivel a k-közép algoritmus nem kezeli jól a zajos adatokat, ezeket az adatokat is mindenképpen besorolja valamelyik klaszterbe, jelentősen torzítva az adott klasztert, a klaszterezés megkezdése előtt szűrnünk kell az adathalmazt. Elsősorban statisztikai jellemzők alapján szűrtük az adathalmaz elemeit, kívülálló (outlier) elemeket keresve. A szűrés segítségével az olyan profilokat is eltávolítottuk az elemzendő adathalmazból, ahol a kattintások száma túl alacsony volt, azaz statisztikai szempontból nem voltak jelentősek.

A k-közép algoritmus bemeneteként meg kell adnunk a kialakítandó klaszterek számát is. Ez, ahogy sok esetben, a mi esetünkben sem ismert. Ezért az alábbi heurisztikát alkalmazzuk: az algoritmust futassuk több klaszterrel és használjuk ki az algoritmus szegmentáló tulajdonságát. Azt tételezzük fel, hogy ha felülbecsüljük az adathalmazban meglévő klaszterek számát, akkor a k-közép algoritmus szegmentálja az elvileg egy klaszterbe tartozó elemeket, azaz felosztja őket nagyon hasonló elemmel jellemezhető klaszterekbe. Ezért a klaszterező algoritmusunk futtatása után még egy újabb lépést is bevezetünk, ami a hasonló klaszterek összevonása lesz. A gyakorlat alátámasztotta a klaszterek számára vonatkozó heurisztikánkat: 10 klasztert adtunk meg a k-közép algoritmus bemeneti paramétereként, majd az algoritmus lefutása után, a hasonló klasztereket összevonva megkaptuk a végleges klaszeretést.

4. Eredmények



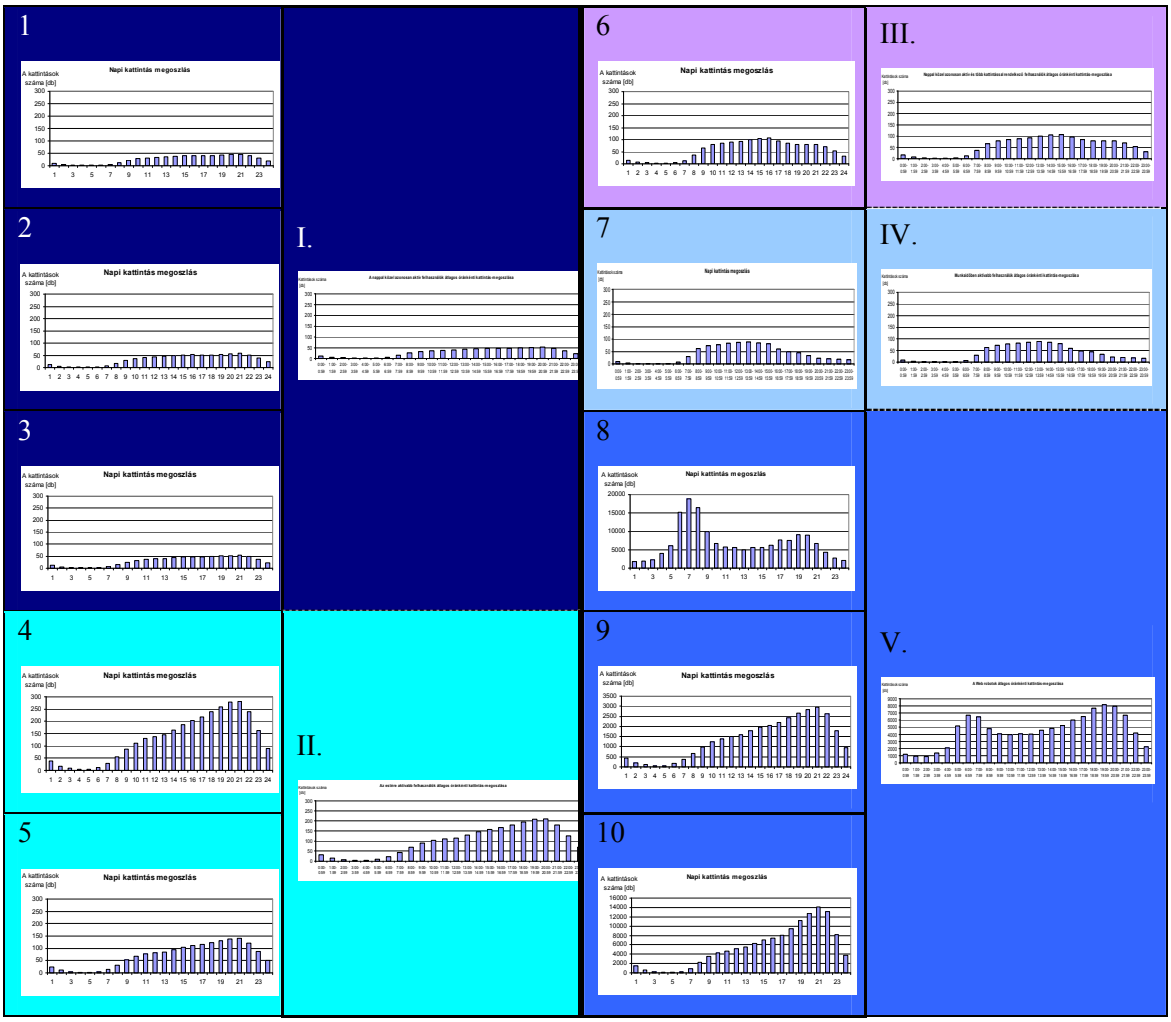
3. ábra: Az in-memory és az out-of-core módszerek futási ideje (bal oldal) és memóriahasználata (jobb oldal)

A *3. ábra* támasztja alá az out-of-core módszerek szükségességét. A valós adatok feldolgozása során mértük a futási időt és a felhasznált memóriát egy olyan környezetben, ahol korlátozott memória állt rendelkezésre. A nyers erő módszere a kattintások aggregálását hajtja végre egy hash alapú tárolót használva, de mindezt úgy, hogy csak az operatív memóriát használja. A futási időt ábrázoló grafikonon jól látható, hogy abban az esetben ha a fizikai memória eléri határait és a virtuális memóriakezelésnek megfelelően kilapozás történik, a futási idő kezelhetetlenül megnő. A felhasznált memóriát ábrázoló grafikonon

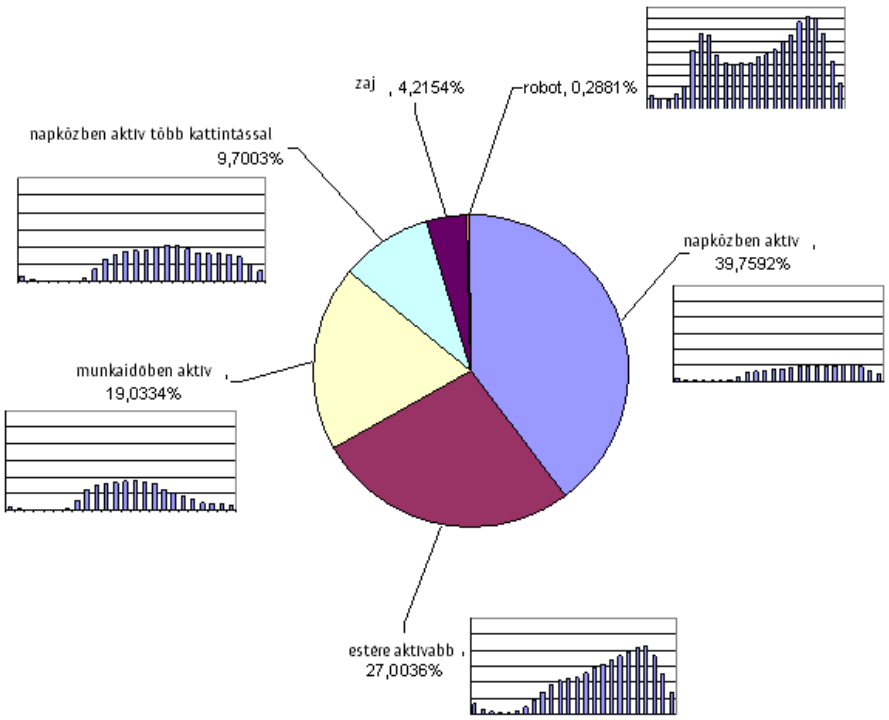
kivehető, hogy míg a in-memory megközelítésben nincs korlát a felhasznált memória mennyiségére az algoritmuson belül, addig a szakaszos feldolgozó algoritmusok szabályozhatóan korlátos memóriaigényűek.

A *4. ábra* a kialakított 10 klasztert láthatjuk, illetve a hasonló klaszterek összevonásával keletkező klasztereket. A klaszterek jellemzése a klaszterben lévő profilok átlagával történik. Látható, hogy a 10 klaszterrel túlbecsültük a klaszterek számát, több olyan klaszterjellemzőt is kaptunk, amelyek hasonlítanak egymásra. Ezért ezeket összevontunk, az új klasztert jellemző elemet pedig úgy határoztuk meg, mint az összevont elemek reprezentáns elemének átlagát.

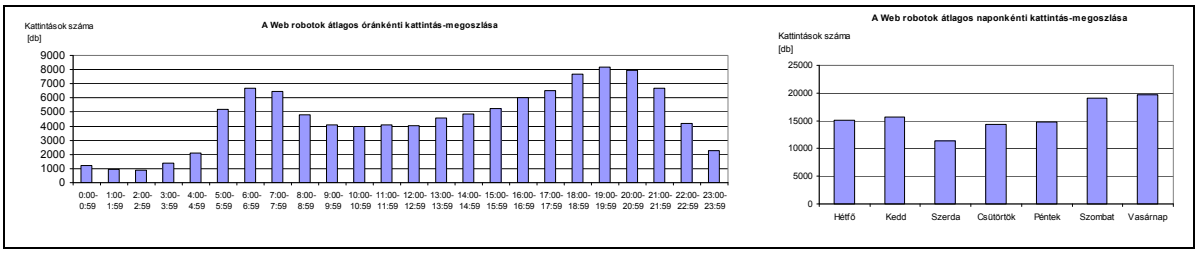
Az *5. ábra* a kialakított klaszterek egymáshoz viszonyított arányait szemlélteti: ez alapján látható, hogy a napközben aktív felhasználók száma a legmagasabb, amit az estére aktívabb felhasználók követnek, majd a munkahelyen internetezők. A sorban az egész nap aktív felhasználók következnek, akiknek nagyobb az aktivitásuk, mint az elsőként említett csoportnak, majd a Web robotok csoportja a legkisebb. Emellett az adathalmaz kb. 4,21%-át a klaszterezés előtt kiszűrtük, mivel kívülálló pontok voltak vagy a rövid időtartamuk miatt nem bírtak jelentőséggel statisztikai szempontból.



4. ábra: A kialakított klaszterek összevonása



5. ábra: A kialakított klaszterek egymáshoz viszonyított arányai



6. ábra: A kialakított klaszterek jellemzése

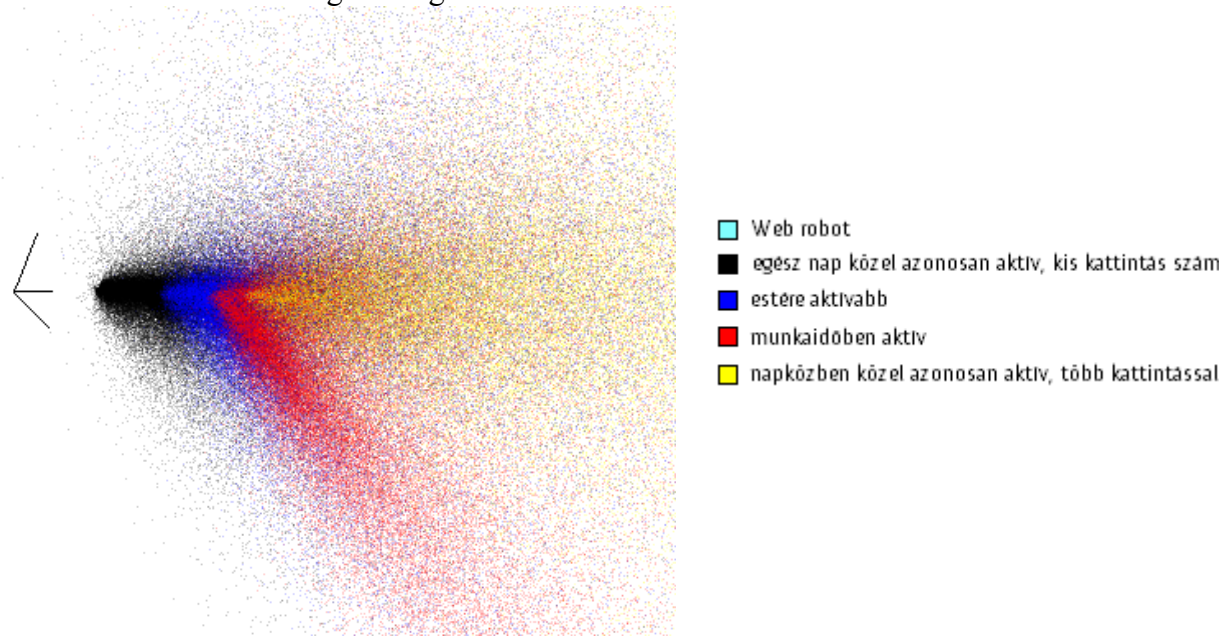
A kialakított tipikus felhasználói csoportok óránkénti és naponkénti átlagos kattintás-megoszlására vonatkozó diagramokat a *6. ábra* szemlélteti. Jól kivehetőek az egyes grafikonok közötti különbségek és hogy ezek hogyan illenek bele az értelmezésbe. Az összehasonlítóság érdekében a robot profilt kivéve azonos skálázás mellett ábrázoltak a különböző grafikonok. Az egy sorban található grafikonok közül a bal oldali szemlélteti a kattintások megoszlását óránkénti-, míg a jobb oldali grafikon a kattintások megoszlását napi bontásban. A továbbiakban az egyes klaszterekre a grafikonjaik sorszáma alapján hivatkozok.

Habár az első és a negyedik sorban levő grafikonok hasonló viselkedést ragadnak meg, a kattintások száma azonban megkülönbözteti őket, amint ez a két ábrán észrevehető. Ezek alapján az első klaszter egy a teljes nap folyamán aktív felhasználói csoportot jelöl, addig a negyedik klaszter egy hasonló viselkedésű, de aktívabb felhasználói csoportot jelöl.

A második grafikon egy, az esti órákban csúcsosodó aktivitást szemléltet, amellett, hogy a nappali aktivitás sem elhanyagolható. Ezek alapján ezt a klasztert úgy értelmezhetjük, mint az estére aktívabb felhasználókat tartalmazó csoportot.

A harmadik grafikonon jól kivehető a munkaidőben történő intenzívebb aktivitás, ez alapján pedig e csoport címkéje a munkahelyi internetezők csoportja lehet. Ezt a címkét mind az óránkénti kattintások száma, mind a naponkénti kattintások száma alátámasztja, mert órás bontásban a munkaidőben csúcsosodik a kattintások megoszlása, míg a heti megoszlást tekintve a hétvégi kattintások száma alacsonyabb, mint a hétközbeli aktivitások száma.

Az ötödik grafikon napi kattintás megoszlását megfigyelve látható, hogy átlagosan minden órában a kattintások száma egy adott korlátnál nem kisebb, ami hosszú távon nem lehet jellemző emberi viselkedés. A kattintások magas száma is alátámasztja, hogy e klaszter automatizált viselkedés ragadt meg.



7. ábra: Az klaszterezett adathalmaz egy PCA alapú 3D megjelenítése

A ***7. ábra*** szemlélteti az adathalmaz egy PCA (Principal Component Analysis, főkomponens analízis) alapú transzformációját a háromdimenziós térbe, míg a különböző színek az egyes klasztereket jelentik. A PCA a sokdimenziós vektor egyes dimenzióinak lineáris kombinációját képezi, annak érdekében, hogy a lehető legtöbb információt a három dimenzióba transzformálhassa. Az ábrán négy szín lehet elkülöníteni, a Web robotok csoportja nem látszik az ábrán, mivel azok számossága a többi klaszterhez képest alacsony.

5. Összegzés, lehetséges továbbfejlesztések

A cikkben két optimalizált előfeldolgozási módszer került bemutatásra, amelyek segítségével az alacsony absztrakciós szintű adathalmazt a megfelelő formátumúra transzformálhatjuk. Mind a szakaszos feldolgozás, mind a k-utas összefésülésen alapuló módszer ún. out-of-core módszer, másodlagos memóriát is használnak a feldolgozásuk során. Az előállt profilok alapján a tipikus felhasználói csoportok meghatározásával értékes tudást nyújthatunk a szolgáltatók számára.

Lehetséges továbbfejlesztési irányokat két részre oszthatjuk: a felhasználói profilok pontosítását jelenti az egyik lehetséges irányt. Amint azt a bevezetőben is említettük, az ún. third-party cookie-k (C3) mellett az ún. first-party cookie-k (C1) is eltárolásra kerülnek egy-egy felhasználói kattintás esetén. Mivel a C1 sütik ritkábban kerülnek törlésre, ezért ezek mentén a különböző C3 sütivel rendelkező, de valójában azonos felhasználóhoz tartozó profilok összevonhatóak. Ez a megközelítés egy out-of-core páros gráf építő eljárás segítségével oldható meg.

A másik lehetséges továbbfejlesztési irányt a klaszterező algoritmus továbbfejlesztése jelentheti. A k-közép egy módosított változata, a k-közép++ [4] az óvatos középpontválasztással próbálja meg gyorsítani klaszterezést és javítani is azt (megelőzni a lokális optimumba ragadását). Ezt alkalmazhatjuk a saját adathalmazunkra is, azonban a középpont választás során figyelembe kell vennünk a nagy adathalmaz okozta problémákat is.

6. Köszönetnyilvánítás

A munka szakmai tartalma kapcsolódik a "Minőségorientált, összehangolt oktatási és K+F+I stratégia, valamint működési modell kidolgozása a Műegyetemen" c. projekt szakmai célkitűzéseinek megvalósításához. A projekt megvalósítását az ÚMFT TÁMOP-4.2.1/B-09/1/KMR-2010-0002 programja támogatja.

Hivatkozások

[1] Bodon, Ferenc. Klaszterezés. Adatbányászati algoritmusok (tanulmány). Budapest.

[2] Fogaras, Dániel, and Lukács, András. Klaszterezés. Informatikai algoritmusok. Vol. 2. Budapest: ELTE Eötvös Kiadó, 2005. 1409 1435

[3] Athur, David and Vassilvitskii, Sergei. How slow is the k-means method? (Previously titled On the Worst Case Complexity of the k-means Method.), Appeared in SoCG 2006 (Sedona, Arizona).

[4] D. Arthur, S. Vassilvitskii: k-means++ The Advantages of Careful Seeding 2007 Symposium on Discrete Algorithms (SODA).

[5] Kleinberg, Jon: An Impossibility Theorem for Clustering, Advances in Neural Information Processing Systems (NIPS) 15, 2002.

[6] Salvatore C. L., Perego O. R. 2006, Mining frequent closed itemsets out-of-core, 6th SIAM International Conference on Data Mining, pp. 419-429.

[7] Nguyen Nhu, S., Orlowska, M. E. 2005, Improvements in the data partitioning approach for frequent itemsets mining, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-05), pp. 625-633.

[8] Nguyen S. N. and Orlowska M. E. 2006, A further study in the data partitioning approach for frequent itemsets mining, ADC '06 Proceedings of the 17th Australasian Database Conference, pp. 31-37.

[9] Benczúr A. A., Csalogány K., Lukács A. Rácz B. Sidló Cs., Uher M. and Végh L., Architecture for mining massive web logs with experiments, In Proceedings of the HUBUSKA Open Workshop on Generic Issues of Knowledge Technologies

[10] Iváncsy, R. and Juhász, S. 2007, Analysis of Web User Identification Methods, Proc. of IV. International Conference on Computer, Electrical, and System Science, and Engineering, CESSE 2007, Venice, Italy, pp. 70-76.Hivatkozás3

[11] Vitter, J. S. Algorithms and Data Structures for External Memory. Boston, MA: Now, 2008. Print

[12] Martin Isenburg, Stefan Gumhold, Out-of-Core Compression for Gigantic Polygon Meshes, Proceedings of SIGGRAPH'03, pages 935-942, July 2003.