Eötvös Loránd Tudományegyetem

Információs Rendszerek Tanszék

Adaptív linkkiemelő internetes böngészéshez

Diplomamunka

Készítette: Gábor Bálint

Eötvös Loránd Tudományegyetem,

Programtervező matematikus hallgató

Témavezető: dr. habil. Lőrincz András

ELTE TTK Információs Rendszerek Tanszék

2004. június 15.

1. Kivonat

Az egyre növekvő méretű weben a friss információk felkutatásának segítése komoly feladat elé állítja a mesterséges intelligenciát használó módszereket. A nagy keresőrendszerek, mivel indexelési ciklusuk legalább egy hónap, ebben csak korlátozottan segíthetnek. A mesterséges intelligencia módszerei viszont lehetőséget nyújthatnak az új oldalak gyorsabb megtalálására. Az általunk javasolt módszer a böngészés közben elérhető linkek közül kiemeli (kiszínezi) azokat, melyek a felhasználó számára érdekesek lehetnek. A kiemelés alapját a felhasználó korábbi linkválasztásai adják.

Módszerünk elemei a következők: Szövegosztályozókat fejlesztettünk ki, melyek egy-egy osztályba tartozó dokumentumokat tudnak felismerni. Az előre betanított osztályozókhoz gyorsan adaptálódó lineáris súlyok tartoznak, ezek fejezik ki, hogy a szövegosztály tartalma mennyire felel meg a felhasználó érdeklődésének. A böngészés során a módszer a súlyok hangolásával tanul rá a felhasználó céljára. Kérdésünk az volt, hogy - a szomszédos dokumentumok letöltése, ezek értékének megbecslése és a felhasználó lépései alapján tanulva - milyen gyorsan és milyen pontosan becsülhető dokumentumok felhasználó szerinti értéke. Háromféle adaptációs eljárást használtunk a súlyok állítására: (i) megerősítéses tanulást (RL), (ii) csúszóablakos technikát és (iii) értékbecslési hibával modulált csúszóablakos technikát. Tesztjeink során a felhasználót egy modellel helyettesítettük. Első lépésben az internetet is modelleztük, majd a legjobb algoritmussal, modell-felhasználót használva, internetes teszteket is végeztünk.

Vizsgálataink alapján módszerünk képes arra, hogy kevesebb, mint 5 lépés alatt rátanuljon a felhasználó céljára, és kiemelje a számára legérdekesebb oldalakat.

Tartalomjegyzék

2. Bevezetés

Napjaink legnagyobb információforrása az Internet. Az Internet kiszolgálóin megtalálható HTML oldalak által alkotott world wide web mérete exponenciálisan növekszik, míg 1995-ben 5 millió oldalt tartalmazott, 2000 elejére elérte az 1 milliárd oldalt, és mostanra megközelítette a 10 millárdot. A weben található új információ ennél is nagyobb mennyiségű, hiszen egyre nagyobb arányban találhatók dinamikus oldalak, melyek tartalma napról napra, sőt óráról órára megújul.

Ekkora információmennyiség használata esetén kritikus kérdéssé válik a keresések hatékonysága. Az interneten alapvetően kétféleképpen tudunk keresni. Egyrészt rendelkezésünkre állnak nagy keresőrendszerek (pl. Altavista, Google), melyek indexelik a weben található tartalmat, másrészt a hiperhivatkozásokat használva magunk is barangolhatunk oldalról oldalra a minket érdeklő témák után kutatva.

Mind a két módszernek vannak hátrányai. A gyors növekedés és a nagy változékonyság komoly feladat elé állítja azokat keresőrendszereket, amelyek a web teljességének indexelését tűzték ki célul. Mire minden oldal tartalmát felviszik adatbázisukba, másfél hónap is eltelik, és eddigre sok oldalnak megváltozik a tartalma. Ezért a friss információk felkutatásának lehetősége az általános keresőrendszerekkel erősen korlátozott. A Google GoogleNews szolgáltatása némileg változtat ezen a helyzeten: előre meghatározott híroldalak, portálok tartalmát, melyekről tudható, hogy fontosak és változékonyak, sűrűbben indexelik. Viszont ha a téma, amelyben friss információk után kívánunk keresni, speciális, viszonylag kevés ember érdeklődésére számot tartó, akkor ez a módszer sem segít.

Másik probléma a keresőrendszerekkel, hogy sokszor nehéz néhány kulcsszóval kifejezni azt, hogy mi az amire tényleg kíváncsiak vagyunk, és ezért hamis, vagy túl sok, akár több százezer találatot kapunk. Bár a találatok a keresőrendszer mértéke

alapján rendezve vannak (ez a Google-nél[1] pl. a PageRank), ez a sorrend nem feltétlenül felel meg a felhasználó érdeklődésének megfelelő sorrendnek. Néhány kulcsszó kifejezőereje sokszor kevés a felhasználó bonyolult, nehezen meghatározható céljának leírására.

Az általános keresőrendszerek problémáinak orvoslására egy új megközelítés a témaspecifikus keresők használata. A témaspecifikus keresők célja nem a teljes web feltérképezése és indexelése, hanem hogy egy konkrét témában minél részletesebb keresési lehetőség megadása. Ennek megvalósításához témaspecifikus barangolókra van szükség, melyek képesek egy adott oldalról eldönteni, hogy beletartozik-e az ő témájukba. Ilyen barangolók segítségével kifejlesztett keresőrendszerekben sokkal kevesebb oldalt kéne indexelni, a működtetés erőforrásigényei jóval alacsonyabbak lennének. Így a saját keresőmotor működtetésének lehetősége jóval szélesebb körben válna elérhetővé, kisebb közösségeknek is lehetne saját keresőjük az őket érdeklő témában.

A diplomamunkában részletezett munka keretében az egyéni, böngészés közbeni keresést segítő mesterséges intelligencia módszert fejlesztettünk ki. A algoritmus lényege, hogy a felhasználó minden lépése után (a korábbi lépések alapján) megpróbálja megbecsülni, hogy az egyes elérhető oldalak mennyire felelnek meg a felhasználó céljának. Ezen információ birtokában kiemelheti azokat a hivatkozásokat, melyek a legértékesebb dokumentumokra mutatnak. A módszer adaptív, a felhasználó minden lépése után módosítja becsléseit az új lépésnek megfelelően.

[1] http://www.google.com

3. Irodalmi áttekintés

3.1. A web szerkezete

Ahhoz, hogy eredményesen tudjunk barangolni a weben, akár az egyéni keresés során, akár intelligens barangolóprogramunkat használva, elengedhetetlen ismernünk a web szerkezetének sajátosságait.

Tekintsük a webet egy irányított gráfnak, ahol a csúcsok az oldalak, az élek az oldalak közötti hivatkozások. Albert és Barabási vizsgálataik során úgy találtak, hogy a web gráfjának fokszámeloszlása – mind a bemenő mind a kijövő éleket tekintve – széles tartományban hatványfüggvény eloszlású [4, 1]. Ezt az eredményt más kutatások is megerősítették [7]. Ennek a fokszámeloszlásnak köszönhetően a web skálamentes szerkezetű.

Albert és Barabási szerint ez a fokszámeloszlás annak köszönhető, hogy az újonnan létrejövő dokumentumok nagy valószínűséggel hivatkoznak olyan oldalakra, melyekre már akkor is sok másik oldal mutatott. Ezen gondolat alapján egy készítettek egy modellt, melyet *preferential connectivity*-nek neveztek el. A modell által generált gráf fokszámeloszlása tényleg hatványeloszlást követett.

A skálamentességen kívül a web gráfja még egy fontos tulajdonsággal rendelkezik: *kis világot* alkot. A kis világ gráfok a következő két tulajdonsággal rendelkeznek: (i) gráf csúcsainak logaritmusával arányos az átmérőjük, (ii) nagy a klaszterezési együtthatójuk. A nagy klaszterezési együttható informálisan azt jelenti, hogy egy csúcs szomszédai nagy valószínűséggel egymással is szomszédosak. Formálisan az i. csúcs klaszterezési együtthatója a következőképpen kaphatjuk meg: $C_i = \frac{2*E_i}{k_i-1)k_i}$, ahol az k_i az i. csúcs fokszáma, E_i az i. csúcs és szomszédai által feszített részgráf élszáma. A teljes gráf klaszterezési együtthatója pedig a csúcsok klaszterezési együtthatóinak átlagaként áll elő.

3.2. Témaspecifikus barangolók

A nagy keresőrendszerek célja, hogy a teljes webet indexeljék, ezért olyan barangolókat alkalmaznak adatbázisaik elkészítésére, melyek a web gráfjának teljes bejárásával próbálkoznak. A témaspecifikus barangolók ezzel szemben csak azokat az oldalakat próbáljak megkeresni, melyek az őt működtetők keresési céljainak megfelelnek [9, 8, 12, 20, 17, 22, 16]. A barangolás során csak azokon a linkeken haladnak tovább, amerre témájukba vágó dokumentumokat találnak. A döntés történhet az oldalak tartalma alapján, ilyenkor a továbbhaladási irányra vontakozó döntés meghozatalához az összes elérhető oldalt le kell tölteni, vagy történhet csak a linkek, esetleg a linkek környezetének szövege alapján. Ez utóbbi esetben nehezebb a döntést meghozni, viszont az eljárás kevesebb letöltést igényel.

Hogyan ismerjék fel a barangolók, hogy mely oldalak relevánsak a keresés szempontjából? Az egyik lehetőség a megerősítéses tanulás használata, hiszen így a kereső felhasználójának csak visszajelzéseket kell adnia a barangolónak, hogy melyek voltak relevánsak számára a talált oldalak közül, és melyek nem. A jó oldalakért a barangoló jutalmat kap, a rossz oldalakért pedig büntetést, így fokozatosan rátanul a felhasználó szándékára. A megerősítéses tanulás módszeréről részletesen olvashatunk [25]-ben, a barangolókban való használatáról pedig a [23, 19] munkákban.

Érdekes megközelítés a keresés irodalmában a környezetérzékeny barangolók használata [8, 12, 17, 19, 15]. Sokszor nehéz megtalálni közvetlenül a jó oldalakat, ezért érdemes lenne érzékennyé tenni a barangolót arra, hogy mikor tartozkódik a keresett téma közeli környezetében. Ehhez szükség van arra, hogy előre feltérképezzük néhány releváns oldal környezetét, amire a rendszert taníthatjuk. Ehhez vezették be a *környezeti gráf* fogalmát. A környezeti gráf egy fa, melynek gyökerében a céldokumentum található, első szintjén olyan dokumentumok vannak, melyekről közvetlenül mutat link a céldokumentumra, második szintjén olyanok, melyekről mutat

link a gráf első szintjén lévő dokumentumta, tehát melyekről két lépésben érhető el a céldokumentum, és így tovább. Ha a barangoló a környezeti gráf egyes szintjeihez hasonló dokumentumokat talál, akkor az jelezheti a keresett témához tartozó dokumentumoktól vett távolságát.

A barangolók hatékonysága szempontjából fontos körülmény a web korábban leírt skálamentes kis világ szerkezete. A sűrűn összekötött klaszterekbe a barangolók könnyen csapdázódnak, hiszen a linkek nagyrésze a klaszteren belülre vezet. Így lemaradhatunk a klaszteren kívüli érdekes oldalakról. Erre megoldást nyújthat, ha egy hosszútávó memóriában eltároljuk azokat az oldalakat, amelyek jó kiindulópontnak bizonyultak a barangolás során, tehát innen indulva gyorsan találtunk jó oldalakat. Ha már régóta nem találunk megfelelő oldalt, mert egy rossz klaszterbe csapdázódtunk, akkor ebből a hosszútávú memóriából tudunk választani már korábban bevált kiinduló oldalt, és onnan folytathatjuk a keresést.

3.3. Szövegreprezentáció

Ahhoz, hogy az Interneten található dokumentumok közötti keresésre mesterséges intelligencia algoritmust adjunk, meg kell határozni az algoritmus állapotterét, a dokumentumok szövegeinek reprezentációját. A mi megoldásunkban a dokumentumokat szövegosztályozókkal értékeltettük ki, és a kimenetekből alkotott vektorok voltak a dokumentumok reprezentációi.

A szövegosztályozókat felügyelt tanulási módszerrel készíthetjük el. Adott egy tanító dokumentumhalmaz, melynek elemei osztályokba vannak sorolva. A tanító halmaz alapján elkészített osztályozók az ismeretlen dokumentumokat megpróbálják beosztani az osztályok valamelyikébe [13].

Az általunk használt szövegosztályozó eljárások nem közvetlenül a szövegeket kapják bemenetként, hanem vektorokat, melyeket a szövegek szózsák alapú repre-

zentációiból nyerünk. A szózsák alapú reprezentáció lényege, hogy a szövegekből kiválasztunk n db szót, ezek lesznek a feature-ök. Minden szöveg leírható egy n-dimenziós vektorral, melynek i-edik eleme az a szám lesz, ahányszor az i-edik feature-szó szerepel a szövegben. Tehát a szózsák alapú reprezentáció esetén a szavak sorrendjét teljesen figyelmen kívül hagyjuk, sok nyelvi információt eldobunk. Ennek ellenére a gyakorlatban sok információkeresési algoritmus kielégítő módon működik ezzel az igen egyszerű eljárással. Nyilvánvaló, hogy az eljárás kritikus pontja a feature-ök kiválasztásának minkéntje. A cél az, hogy a feature-ök minél inkább jellemezzék a dokumentumokat. Ha túl gyakori szavakat választunk, akkor azok jelentése valószínűleg túl általános lesz, ezért keveset mondanak az egyes dokumentumok témájáról. A túl ritka szavak mellékesek lehetnek, nem lényegesek, sőt az is lehet, hogy csak egy elgépelésnek köszönhetik létüket. A feature-ök számának megválasztása szintén sarkalatos kérdés: a túl kevés feature nem rendelkezik elegendő kifejezőerővel, túl sok feature viszont zajt vihet a rendszerbe.

Joachims [13]-ben leírt eljárását használtuk a feature-ök kiválasztására. Az eljárás három lépésből áll:

1. túl ritka szavak kiszűrése
2. túl gyakori szavak kiszűrése
3. azon szavak kiválasztása melyek a legnagyobb kölcsönös információval rendelkeznek a szövegosztályokra vonatkozóan

A kölcsönös információ azt méri, hogy mennyiben változtatja meg egy véletlen vál-

tozó ismerete egy másik véletlen változó eloszlásának entrópiáját [11].

$$
\begin{aligned}
I(T,w) &= E(t) - E(t|w) \\
&= -\sum_{C\in\mathcal{C}} Pr(T(d)=C) log(Pr(T(d)=C) \\
&+ \sum_{C\in\mathcal{C}} Pr(T(d)=C, w=0) log(Pr(T(d)=C|w=0) \\
&+ \sum C \in \mathcal{C} Pr(T(d)=C, w=1) log(Pr(T(d)=C|w=1)
\end{aligned}
$$

ahol $Pr(T(d) = C)$ annak a valószínűsége, hogy egy véletlenül választott d dokumentum a C osztályba tartozik. A $Pr(T(d) = C, w = 1)$ illetve a $Pr(T(d) = C, w = 1)$ annak a valószínűsége, hogy egy d dokumentum a C osztályba tartozik, és megtalálható, illetve nem található meg benne a w szó. Hasonlósan értelmezett $Pr(T(d) = C|w = 1)$ és $Pr(T(d) = C|w = 1)$, csak feltételes valószínűségekkel.

A ritka és gyakori szavak kiszűrése után tehát azokat a szavakat választjuk feature-nek, melyekhez a legnagyobb $I(T, w)$ értékek tartoznak.

A következőekben háromféle szövegosztályozó eljárást mutatunk be: a TFIDF, a Naive Bayes és az ezekre épülő PrTFIDF osztályozót.

3.3.1. TFIDF osztályozó

A TFIDF (term frequency inverse document frequency, azaz szófrekvencia inverz-dokumentumfrekvencia) osztályozó a dokumentumokat reprezentáló vektorokban $TF \cdot IDF$ értékeket tárol. Egy d dokumentumhoz és egy w feature-szóhoz tartozó $TF(w, d)$ (szófrekvencia) érték a w szó előfordulásainak száma d dokumentumban. Az $IDF(w)$ (inverz-dokumentumfrekvencia) a $DF(w)$ érték segítségével számolható, mely a w szót tartalmazó dokumentumok számát jelenti: $IDF(w) = log(\frac{|D|}{DF(w)}$, ahol $|D|$ a dokumentumok száma. A d dokumentumot reprezentáló

$d = (d^{(1)}, d^{(2)}, \ldots, d^{(|F|)})$ vektor i-edik komponensét tehát a következőképpen számoljuk: $d^{(i)} = TF(w_i, d) \ldots IDF(w_i)$.

Az algoritmus minden C osztályhoz egy prototípus vektort rendel: $c = \sum_{d \in C} d$, egy ismeretlen dokumentumhoz az osztályozó abba az osztályba fogja sorolni, melynek prototípus vektorával a d' reprezentáló vektor a legkisebb szöget zárja be:

$$H_{TFIDF}(d') = argmax_{C \in \mathcal{C}} \frac{d' \times c}{||d'|| \times ||c||} \tag{1}$$

3.3.2. Naive Bayes osztályozó

A naive Bayes osztályozó egy valószínűségi modellt használ a szövegek keletkezésére. A modell szerint a dokumentumok úgy keletkeznek, hogy szavakat generálunk osztályonként külön-külön valószínűségeloszlások alapján, és egy új szó generálása független az addig előállított szavaktól. Ez a feltétel természetesen a valóságban nem teljesül, de konzisztens a szózsák alapú reprezentációval, és segítségével egyszerűen készíthetünk jól működő osztályozókat.

Az osztályozó azt próbálja megbecsülni, hogy mekkora a valószínűsége, hogy a d' dokumentum a C osztályban van ($Pr(C|d')$). Az osztályozó kimenete a d' dokumentumra:

$$H_{BAYES}(d') = argmax_{C \in \mathcal{C}} Pr(C|d') \tag{2}$$

A Bayes-tétel segítségével a következő átalakítást végezhetjük:

$$Pr(C|d') = \frac{Pr(d'|C)Pr(C)}{\sum_{C' \in \mathcal{C}} Pr(d'|C')Pr(C')} \tag{3}$$

Tehát a $Pr(C)$ és $Pr(d'|C)$ értékeket kell megbecsülnünk. A $Pr(C)$ becslése egyszerű, a C osztályba tartozó tanító dokumentumok száma osztva az összes tanító dokumentum számával, tehát $\widehat{Pr}(C) = \frac{|C|}{\sum_{C' \in \mathcal{C}} |C'|}$.

Feltételeztük, hogy egy szó előfordulásának valószínűsége csak attól függ, hogy a dokumentum melyik osztályba tartozik, és független a dokumentum többi szavától.

Ez alapján a $Pr(d'|C) = \prod_{i=1}^{|d'|} Pr(w_i|C)$, ahol a w_i a d' dokumentum i-edik olyan szava, amely szerepel a feature-ök között. Tehát a dokumentumok osztálybeli előfordulásának valószínűségét visszavezettük szavak előfordulásának valószínűségére. A w_i szó C osztálybeli előfordulásának valószínűségét pedig így becsüljük:

$$Pr(w_i|C) = \frac{1 + TF(w_i, C)}{|F| + \sum_{w' \in F} TF(w', C)} \tag{4}$$

ahol F a feature szavak halmaza.

Ezt visszahelyesítve a $Pr(d'|C)$ képletébe, kombinálva 3-al és 2-vel megkapjuk az osztályozó döntési függvényét.

3.3.3. PrTFIDF osztályozó

A valószínűségi TFIDF (PrTFIDF) osztályozó tervezésekor különbséget teszünk egy dokumentum és annak reprezentációja között. A szöveg reprezentációját a Θ valószínűségi leképezés határozza meg, ezután az osztályozó a reprezentáció alapján kategorizálja a dokumentumot. A döntés az előző esethez hasonlóan most is valószínűségi.

$$H_{PrTFIDF}(d') = argmax_{C \in \mathcal{C}} Pr(C|d', \Theta). \tag{5}$$

$Pr(C|d', \Theta)$-t két részre bonthatjuk.

$$Pr(C|d', \Theta) = \sum_{x} Pr(C|x) Pr(x|d', \Theta) \tag{6}$$

ahol $Pr(x|d', \Theta)$ annak valószínűsége, hogy a d' dokumentum reprezentációja x Θ leképezés esetén, $Pr(C|x)$ pedig annak valószínűsége, hogy az x repreznetációjú dokumentum a C osztályba tartozik.

A most bemutatásra kerülő Θ leképezést a TFIDF osztályozó módszere motiválta. A dokumentumokat egyetlen szó fogja reprezentálni, de a ennek a szónak a kiválasztása valószínűségi alapon fog történni. Annak valószínűsége, hogy egy szó

képezi a reprezentációt, megegyezeik annak a valószínűségével, hogy őt választjuk ki a dokumentum szózsákjából.

$$Pr(x|d', \Theta) = Pr(w|d', \Theta) = \frac{TF(w, d')}{\sum_{w' \in F} TF(w', d')} \tag{7}$$

$Pr(C|x)$-et megint két részre bonthatjuk a Bayes-tétel alapján.

$$Pr(C|x) = Pr(C|w) = \frac{Pr(w|C)Pr(C)}{\sum_{C' \in \mathcal{C}} Pr(w|C')Pr(C')} \tag{8}$$

$Pr(w|C)$ a tapasztalati eloszlás most is becsülhető, ismét feltéve a szavak előfordulásának függetlenségét.

$$\widehat{Pr}(w|C) = \frac{1 + TF(w, C)}{|F| + \sum_{w' \in F} TF(w', C)} \tag{9}$$

$Pr(C)$-t is a már ismertetett módon becsüljük ($\widehat{Pr}(C) = \frac{|C|}{\sum_{C' \in \mathcal{C}} |C'|}$).

Ezek alapján az osztályozó döntési függvénye a következő.

$$H_{PrTFIDF}(d') = argmax_{C \in \mathcal{C}} \sum_{w \in F} \frac{Pr(w|C)Pr(C)}{\sum_{C' \in \mathcal{C}} Pr(w|C')Pr(C')} Pr(w|d', \Theta) \tag{10}$$

Megjegyezzük, hogy mi az osztályozó fogalmát az előbb bevezetettől kicsit eltérően fogjuk használni. Látható, hogy a dokumentumokhoz az osztályozó először egy $|\mathcal{C}|$ hosszú vektort rendel, majd a döntési függvénye azt az osztályt választja ki, melyhez a legmagasabb értékű komponens tartozik. Mi nem használjuk ezt a döntési függvényt, hanem közvetlenül a kimeneti vektort tekintjük, mert számunkra külön-külön az összes osztályba tartozás értéke az érdekes. Ezért nem is az egész rendszert hívjuk osztályozónak, hanem minden osztályba való tartozásról mondott véleményt egy külön osztályozó kimenetének tekintjük, így az előbbi osztályozót $|C|$ db osztályozónak tekintjük, az egészet pedig osztályozórendszernek hívjuk.

3.4. Szövegklaszterezés

A szövegklaszterezési feladat lényege, hogy egy nagy strukturálatlan dokumentumhalmazt bontsunk kategóriákra, és ezek a kategóriák a dokumentumok témái alapján

egymástól minél elkülönöltebbek, a kategóriákon belüli dokumentumok pedig minél hasonlóbbak legyenek. Mindezt mi külső tudás nélkül, felügyelt tanulási eszközökkel szeretnénk megoldani. Az ily módon kapott kategóriákat klasztereknek nevezzük. A klaszterekre betaníthatunk osztályozókat, így az új, ismeretlen dokumentumokat is be tudjuk sorolni a klaszterek valamelyikébe.

3.5. Boley klaszterező eljárása

3.5.1. Áttekintés

Boley klaszterező eljárása (Főkomponens alapú partícionáló eljárás, Principal Component Divisive Partitioning) [5] a dokumentumokat TF értékeket tartalmazó feature vektorokként reprezentálja. A vektorokat oszlopvektornak tekintjük, és normáljuk oly módon, hogy az Euklideszi normájuk 1 legyen. Így a $d^T = (d_1, d_2, \ldots, _n)$ vektor értékeire: $d_i = \frac{TF(w_i)}{\sqrt{\sum_j TF(w_j)}}$.

Az eljárás „felülről-lefelé" (top-down) módon osztja kategóriákba a dokumentumhalmazt. Először minden dokumentum ugyanabba a klaszterbe (partícióba) tartozik, melyet a következő lépésben két részre bontunk, az így kapott klasztereket az ezutáni lépésben szintén kisebb klaszterekre bontjuk, és így tovább. Az eljárást egy bináris fával szemléltethetjük, melynek csúcsai klaszterek, egy csúcs gyerekei a neki megfelelő klaszter részklaszterei. A fa leveleiben a tovább már nem bontott klaszterek szerepelnek. Az így felépített fát PDDP fának nevezzük.

Az algoritmus két paraméterében vár pontosításra: (i) milyen módon osztjuk két részre a klasztereket, és (ii) mi alapján választjuk ki a felosztásra kerülő klasztereket.

3.5.2. Klaszterek felosztása

A dokumentumhalmazt egy M mátrixszal reprezentáljuk, melynek oszlopvektorai a dokumentumok feature vektorai: $M = (d^{(1)}, d^{(2)}, \ldots, d^{(m)})$. Egy partíciót egy $M_p = (d^{(1)}, d^{(2)}, \ldots, d^{(p)})$ mátrix ábrázol, melynek oszlopai a partícióba tartozó dokumentumokat reprezentáló vektorok. Ezek természetesen az M mátrix oszlopvektorai közül kerülnek ki, de általában nem egymás utáni oszlopok, bárhol szerepelhetnek. Az M_p mátrix főkomponensei a C kovariancia mátrix sajátvektorai. Legyen $w = \frac{M_p e}{p}$, ahol $e = (1, 1, \ldots, 1)^T$, a partícióba tartozó dokumentumok feature vektorainak átlaga, $C = (M_p - we^T)((M_p - we^T)^T$ a kovariancia mátrix. A Karhunen-Loeve transzformáció a k legnagyobb sajátértékhez tartozó sajátvektorok által kifeszített térbe vetíti az M_p mátrix oszlopait. Eredményként a dokumentumok k-dimenziós ábrázolását kapjuk, az eredeti n-dimenzió helyett. A dimenziócsökkentés az információmennyiség csökkentése mellett – megfelelő k érték esetén – zajtalanítással is jár. A mi esetünkben k értéke 1 lesz, tehát csak a legnagyobb sajátértékhez tartozó sajátvektorra, a főkomponensre vetítünk, melyet u-val fogunk jelölni.

A d_i dokumentumok így kapott projekcióit jelöljük v_i-vel. A dokumentumok a hozzájuk tartozó v_i értékek alapján kerülnek a két új partícióba, $v_i \geq 0$ esetén a dokumentumot az első, $v_i < 0$ esetén a második partícióba helyezzük.

3.5.3. Felosztandó klaszterek kiválasztása

A végeredményként kapott klaszterek nagyban függnek attól, hogy az eljárás során milyen módon választjuk ki a felosztásra váró klasztereket. Egy lehetséges módszer, hogy minden körben minden partíciót felbontunk, így a PDDP fa mindig teljes lesz. Így viszont a levelekben megjelenő klaszterek kiegyensúlyozatlanok lesznek, néhány nagy és sok kicsi klaszterrel, köztük egy dokumentumból állókkal is.

A megfelelő partíciók kiválasztásához bevezetünk egy új mérőszámot, a *szóródottságot*. Minden lépésben a legszóródottabb partíciót osztjuk ketté, ahol a szóródottságot az $A = M_p - we$ mátrix Frobenius normájaként defíniáljuk.

3.6. Információszűrő ágensek

Az utóbbi évtizedben több olyan módszert is kidolgoztak, melynek célja a felhasználók adaptív segítése a böngészés közben. Ezek több szempontból is csoportosíthatóak:

A program futásának helye:

- Kliensoldalon: a felhasználó személyes programjaként működik, a böngészés helyétől függetlenül bármikor segítheti [18, 10, 3].
- Szerveroldalon: egy portál, webes áruház látogatóinak ajánl megfelelő linkeket vagy termékeket az oldalon [24, 26].

Az adaptáció alapja:

- Felhasználói viselkedés alapján: a program nem kap direkt megerősítés, a felhasználó lépései alapján próbál következtetni az érdeklődésére. Egyéb információkat is felhasználhat, például az oldalak olvasásával töltött időt [18].
- Explicit megerősítés: a felhasználónak jeleznie is kell, hogy a látogatott oldal megfelel-e keresése céljának [10, 14].

A kiszolgált felhasználók száma:

- Egy felhasználós rendszer: a program egy felhasználó preferenciáira tanul rá, a tanulásának alapja az oldalak tartalma [18, 10].

- Több felhasználós rendszer: általában a szerveroldali módszereknél jellemző. Az ilyen rendszerek az egyes felhasználóknál szerzett tapasztalataikat más felhasználóknál is hasznosítják. Például, ha egy bizonyos könyv vásárlói általában egy másik könyv iránt is érdeklődnek, akkor az új vásárlókra is igaz lehet ez a kapcsolt preferencia. Az ilyen rendszerek általában felhasználói profilokat tárolnak, és ezekbe próbálják bekategorizálni az új felhasználókat [14, 24]. A tanulás alapja tehát ebben az esetben nem csak az oldalak tartalma, hanem a korábbi felhasználók viselkedése is.

Az oldalakat a módszerek általában TFIDF reprezentációban tárolják, és sokszor a felhasználói profilokat is. Ilyen esetekben az oldalak rangsorolása egyszerűen a TFIDF vektoruk és a felhasználó vektora által bezárt szög alapján történik [10, 14].

A következőekben néhány módszer áttekintése olvasható.

Chen WebMate nevű ágense [10] TFIDF vektorokat használ a felhasználói cél és a dokumentumok reprezentációjára is. Feltételezi, hogy a felhasználónak több célja is lehet, a hozzájuk tartozó vektorokat egymással párhuzamosan tárolja.

A módszer a következőképpen működik. Az elején le kell rögzítenünk, hogy hány felhasználói célt kívánunk kezelni a rendszerrel, nevezzük ezt N-nek. Ha a felhasználó egy oldalt érdekesnek tart, pozitív megerősítést küld a rendszernek. Ekkor ha az N tárolható felhasználói cél még nincs kitöltve, tehát a rendszer még nem kapott ennyi visszajelzést, akkor az oldal TFIDF vektorát elhelyezi a célok között. Ha már mind az N cél foglalt, akkor kiválasztja azt, amellyel az oldal TFIDF vektora a legkisebb szöget zárja be. Úgy tekintjük, hogy a felhasználó ehhez a céljához talált egy új oldalt, és ezért a célt módosítjuk: az új oldal vektorát hozzáadjuk. A betanított rendszert hírportálok tartalmának szűrésére használták.

Liebermann Letizia nevű intelligens ágense [18] a felhasználó korábbi lépései alapján módosítja a felhasználói célról alkotott modellt, mely egyszerűen kulcssza-

vakból áll. Miközben a felhasználó olvas egy oldalt, a rendszer szélességi kereséssel elkezdi felderíteni a környéket, és a relevánsnak tartott oldalakat (melyekben szerepelnek a kulcsszavak) rögtön felajánlja a felhasználónak.

Joachims WebWatcher [14] nevű ágense egy olyan szerveroldali program, amelyik egyszerre felhasználók egy csoportjának nyújt segítséget. A felhasználók az ő segítségével kereshetnek egy téma után, majd a keresés végén jelezniük kell, hogy megtalálták-e az őket érdeklő információt. A rendszer összegyűjti a felhasználók érdeklődési köreit, keresési útvonalait és a visszajelzéseiket. Új felhasználó esetén ellenőrzi, hogy találkozott-e már olyan esettel, amikor egy felhasználó hasonló érdeklődésű volt, ezen az oldalon állt, és a végén sikeresnek minősítette a keresést. Ha talál ilyet, akkor az ő útvonala alapján ajánl linkeket. A felhasználói célokat és az oldalakat is TFIDF reprezentációban tárolja, és többféle tanulási módszert (pl. megerősítéses tanulást) is alkalmaz.

Bailey módszerének [2] lényege, hogy az oldalakon található fontos szavakhoz kézzel létrehozott linkadatbázisokból linkeket rendel. A fontos szavak kiválasztása a felhasználó korábbi lépései alapján történik.

Menczer IntelliShopper [21] ágense internetes vásárlásban tud segítséget nyújtani. Egy szerveren össze vannak gyűjtve a különböző webes áruházak ajánlatai, a rendszer ezek kínálatából tud ajánlatokat tenni. A termékekhez feature-öket rendel (pl. ár, szín, stb.), a felhasználókat pedig a feature-ökhez rendelt súlyok modellezik. A rendszer nem igényel explicit megerősítést: ha a felhasználó eltávolít egy árut a listából, az erős negatív visszajelzésnek minősül, ha egyszerűen továbbgörgeti, az gyenge negatív visszajelzésnek számít. Ha a felhasználó ráklikkel az árura, azt gyenge pozitív, a vásárlást pedig erős pozitív visszajelzésként értelmezi a rendszer.

Az Alexa rendszer[2] egy kereskedelmi termék, mely a Netscape böngészők „What's

[2] http://www.alexa.com

related" funkciójából ismerhető. A böngészőt használók adatait gyűjtve (az egyes oldalakon mennyi időt töltöttek, mely linkeket követték) próbál következtetni az oldalak közötti kapcsolatokra. Ha egy oldal látogatásakor ráklikkelünk a „What's related" gombra, akkor ehhez az oldalhoz hasonlóakat ajánl nekünk a webről. A program algoritmusa zárt, nem hozzáférhető.

Látható, hogy a különböző módszereket igen nehéz összehasonlítani, mivel általában nem is pontosan ugyanazokat a feladatokat oldják meg, ráadásul a tesztelésük módszerei is igen különbözőek.

4. Módszer

Célunk tehát az volt, hogy olyan módszert fejlesszünk ki, mely a felhasználót az egyéni internetes keresésben, böngészésben segíti oly módon, hogy az aktuálisan elérhető linkek közül valamilyen módon kiemeli azokat a linkeket, melyek – a felhasználó korábbi lépései alapján következtetve – a felhasználó érdeklődésére valószínűleg számot tartanak. Ennek érdekében a rendszerünk minden lépésben minden elérhető oldalnak megbecsüli az értéket, és a legértékesebbnek becsült oldalakhoz tartozó linkeket emeli ki. A kiemelés módja a gyakorlatban lehet például az, hogy a hozzá tartozó link más színnel jelenik meg a böngészőben.

4.1. Osztályozók és értékbecslés

Az eljárásunk a következő elemekből épül fel:

- előre betanított osztályozók ("szakértők"), ezek alkotják a rendszer hosszútávú memóriáját;
- az osztályozókhoz tartozó gyorsan adaptálódó relatív súlyok, ezek alkotják a rövidtávú memóriát.

Az osztályozóink PrTFIDF típusú szövegosztályozók, döntési felületük igen éles. Az osztályozók kimenete $+1$ és -1 közötti valós szám lehet, ahol $+1$ jelenti ezt, hogy a bemenetként kapott dokumentum beletartozik a szövegosztályba, -1 azt, hogy nem, a közbülső értékeket köztes „véleményeknek” tekinthetjük. A mi letöltött Geocities adatbázisunkban körülbelül az osztályozókimenetek 5%-a esett $(-0.9, +0.9)$ intervallumba, tehát a maradék 95% mind a $+1$ vagy -1 közelében volt. Az osztályozók minden dokumentumot kiértékelnek, az így kapott vektort használjuk az oldalak reprezentációjára.

Az osztályozókhoz lineáris súlyokat rendelünk, melyek azt fejezik ki, hogy az adott osztályozó mennyire felel meg a felhasználó céljának, érdeklődésének. Lehetséges, hogy az osztályozó hasonló dokumentumokat tart jónak, mint a felhasználó, ekkor pozitív súly jár neki, lehet hogy azok az oldalak tartoznak hozzá, melyek a felhasználót nem érdeklik, ilyen esetben negatív súlyt kell kapnia, és végül előfordulhat, hogy az osztályozó kimenete indifferens a felhasználó választásai szempontjából, ekkor a hozzá tartozó súlynak nullának kell lennie. A súlyok megfelelő beállítása esetén az osztályozók segítségével elkészíthetjük a dokumentumok értékbecslését.

Feltételeztük, hogy a felhasználó keresésének célja modellezhető osztályozók egy lineárisan súlyozott együttesével. Megengedtük, hogy ez a cél a keresés során bejelentés nélkül megváltozzon. Megjegyezzük, hogy feltételezésünk elvben igen nagyszámú cél modellezését teszi lehetővé. Becslésként korlátozzuk a súlyok értékhalmazát ± 1-re. Ekkor 50 osztályozó esetén a lehetséges célok száma 2^{50}. Természetesen nem osztályozók számának növelése a cél, hanem hogy minél alkalmasabb terét alkossák a lehetséges keresési céloknak. Továbbá kérdés az, hogy a fent említett elméleti maximális számú lehetőség közül – konkrét osztályozóhalmaz esetén – mennyi az, amely valóban értelmes keresési szándékot fejez ki.

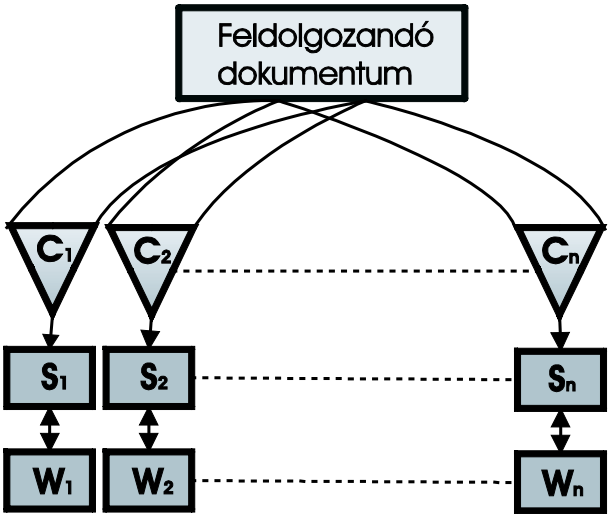
Az adaptációs algoritmus a következőképpen működik. A rendszer fix, előre betanított osztályozókkal dolgozik, melyekhez folyamatosan adaptálódó súlyok tartoznak. Minden lépésben letölti az összes elérhető dokumentumot, megbecsüli az értéküket, majd – a felhasználó következő lépése alapján – frissíti a súlyokat.

Az osztályozók egy dokumentumra adott kimenetét osztályozóvektornak ($\mathbf{S}^T = (S_1, \ldots, S_n)$), a felhasználó célját modellező súlyok aktuális értékeiből alkotott vektort pedig súlyvektornak ($\mathbf{W}^T = (W_1, \ldots, W_n)$ nevezzük. A dokumentum értékbecslése a hozzá tartozó osztályozóvektor és az adott időpillanatbeli súlyvektor skaláris

szorzata (1. ábra). Formálisan:

$$V(S) = \sum_n W_n S_n = \mathbf{W}^T \mathbf{S} \tag{11}$$

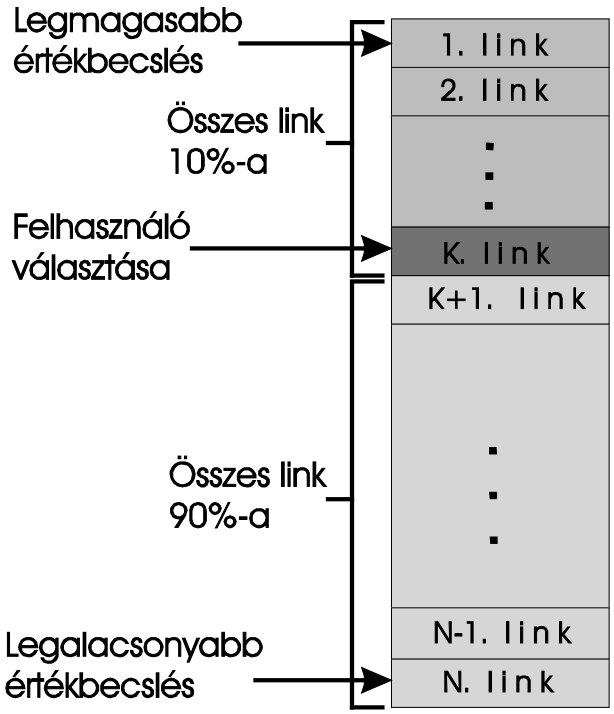
A legnagyobb értékbecslésű linkeket tekintjük kiemeltnek. Célunk az, hogy az oldalak értékbecslése minél inkább hasonlítson a felhasználó értékelésére.



1. ábra. **Értékbecslés**

Az n darab különböző osztályozó $(C_i, i = 1, \ldots, n)$ az új oldalakat kiértékeli, így kapjuk meg az osztályozóvektort $(\mathbf{S}^T = (S_1, \ldots, S_n))$. A dokumentum értékbecslése: $V(dokumentum) = \mathbf{S}^T \mathbf{W}$.

A kísérleteink elemzéséhez be kell vezetnünk egy mérőszámot az értékbecslés hatékonyságának jellemzésére. Azt mondjuk, hogy az értékbecslés 90%-os, ha a felhasználó által ténylegesen választott dokumentum értékbecslésénél az összes dokumentum 10%-a kapott magasabb vagy azonos értékbecslést és 90%-a alacsonyabbat (2. ábra).



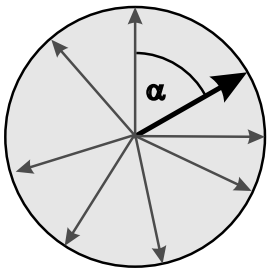
2. ábra. **Értékbecslés hatékonyságának mérése**

90%-os értékbecslés esetén az elérhető dokumentumok 90%-a kap a ténylegesen legjobb dokumentumnál alacsonyabb értékbecslést.

4.2. Felhasználómodell

A kísérleteink során nem valódi (emberi) felhasználókkal, hanem egy felhasználómodellel dolgoztunk. A felhasználómodell osztályozókból és egy súlyvektorból áll. A felhasználómodell (a linkkiemelő rendszerhez hasonlóan) kiértékelteti az elérhető dokumentumokat az osztályozóival, majd a súlyvektor és az osztályozóvektor skaláris szorzataként meghatározza az oldalak értékeit. Mindig a legmagasabb értékű oldalra lép tovább (3. ábra).

Bár a linkkiemelő értékbecslésének eljárása megegyezik a felhasználómodell ér-



3. ábra. **Modell felhasználó**

A W és S vektorok egységhosszúságúra vannak normálva. A fekete vektor ábrázolja felhasználó orientációját, a szürke vektor egy szomszédos dokumentum osztályozóvektorát (a rajz kedvéért csak két dimenzióban). A dokumentum értéke $\cos\alpha$, ahol α a két vektor által bezárt szög.

tékelésével, osztályozóik különbözhetnek, és a linkkiemelő természetesen semmit nem tud a felhasználómodell szerkezetéről.

A valódi életben gyakran előfordul, hogy a böngésző felhasználó témát vált, és más oldalak után kezd keresgélni. Módszerünktől elvártuk, hogy egy bejelentés nélkül célváltozás után gyorsan újra adaptálódjon az új célhoz. Ennek ellenőrzésére bizonyos tesztekben a felhasználómodell súlyvektorát időnként véletlenszerűen megváltoztattuk.

A felhasználómodell és linkkiemelő rendszer osztályozói szempontjából két csoportra oszthatjuk a kísérleteket:

1. a felhasználómodell és linkkiemelő rendszer azonos osztályozókat használ;

2. a felhasználómodell és linkkiemelő rendszer különböző osztályozókat használ.

Az első esetben a feladat arra egyszerűsödik, hogy a linkkiemelő súlyvektora minél

jobban közelítse meg a felhasználó súlyvektorát. A második esetben a feladat nehezebb, ekkor azt várjuk, hogy a linkkiemelő az osztályozóinak segítségével keverje ki ugyanazt a célt, mint ami a felhasználóé volt. Ez természetesen csak akkor lehetséges, ha a rendszer osztályozóhalmaza egyáltalán elvben képes kifejezni a felhasználómodell célját. Mivel algoritmusunk fejlesztésekor a távolabbi cél természetesen az emberi felhasználóval való együttműködés volt, és az emberi felhasználótól nem várhatjuk, hogy célját a linkkiemelő osztályozói közül válassza ki, ezért ez utóbbi tesztek eredménye igen fontos a gyakorlati alkalmazhatóság szempontjából. Úgy is fogalmazhatunk, hogy míg az első típusú tesztek inkább az adaptációs algoritmusok ellenőrzésére alkalmasak (ha egy algoritmus ezen sem működik, akkor biztosan nem alkalmazható a gyakorlatban), addig a második típusúak az osztályozók súlyozott együttesének kifejezőerejét is vizsgálják.

4.3. Felhasznált adatbázisok

A felhasznált adaptációs algoritmusok tesztelésére először egy mesterséges modelladatbázist használtunk. Második lépésben az Internet egy részét letöltöttük, és azon vizsgáltuk a módszer működését a legjobbnak bizonyult adaptációs algoritmussal. Végül az Interneten végeztünk kísérleteket, többféle osztályozóhalmazzal és felhasználómodellel.

4.3.1. Mesterséges modell

A mesterséges modellünkben minden dokumentumot egy 50-dimenziós véletlen számmal reprezentáltunk, mely egy fiktív osztályozóhalmaz kimenetének tekinthető. A véletlen szám komponenseit a $-1, +1$ halmazból választottuk, hozzáadva egy normál eloszlású, 0 várható értékű, 0.1 szórású véletlen számot. Minden lépésben generáltunk 100 ilyen modelldokumentumot, ezek voltak az elérhető oldalak. A felhasználó-

modell ugyanezeket a fiktív dokumentumokat látta, tehát úgy tekinthető, mintha ugyanazokkal az osztályozókkal dolgozott volna, mint a linkkiemelő. A modellt és rajta az algoritmust Matlabban implementáltuk.

4.3.2. A Geocities adatbázis

A Geocities internetes tartomány egy körülbelül 1.5 GB méretű, 90,000 oldalt tartalmazó, egybefüggő részét letöltöttük. Mivel az oldalakon lévő linkek jelentős része nem kerülhetett letöltésre, ezért oldalanként átlagosan csak három link volt elérhető. A legtöbb működő linket tartalmazó oldalon 150 link volt található, míg jelentős számú oldalon egy link sem volt elérhető. A letöltött adatbázis gráfjának fokszámeloszlása hatványeloszlást követett, mint ahogy azt a web egészének vizsgálatakor tapasztalt adatoknál is láttuk. A fokszámeloszlás vizsgálatához használt adatokat az adatbázis 20,000 véletlenszerűen választott dokumentum vizsgálatával nyertük.

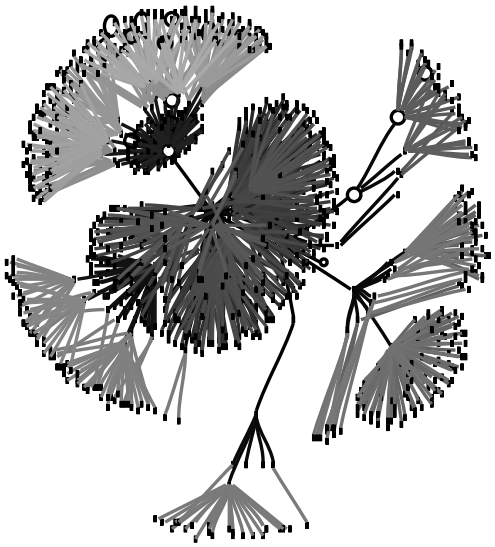
A letöltött adatbázis kezdőoldaláról elindítottunk egy szélességi kereséssel működő barangolót. A felderített rész struktúrája látható a 4. ábrán.

A linkkiemelő algoritmus programját Java-ban készítettük el, ugyanezt a programot használtuk az internetes tesztekhez is.

4.3.3. Internetes tesztek

A linkkiemelő tesztelését az Interneten is elvégeztük. A kezdőoldal a Geocities főoldala volt, a későbbiekben semmilyen korlátozást nem tettünk a böngészés irányára, a felhasználómodell az egész Interneten barangolhatott.

A Geocities és az internetes tesztek során feltételeztük, hogy a felhasználó számára rendelkezésre áll egy 'Vissza' gomb a böngészőben, és/vagy egy 'Előzmények' szolgáltatás, tehát minden korábban látott linket minden lépésben elérhetőnek tekintettünk.



4. ábra. **A Geocities adatbázis struktúrája**

10000 szélességi kereséssel talált dokumentum relatív pozíciója a Geocities adatbázisban.

4.4. Klaszterezés és osztályozás

A Geocities adatbázis letöltött részét (kb. 90,000 html dokumentum) 50 klaszterre bontottuk Boley klaszterezési eljárásával. A klaszterekben lévő dokumentumok osztályozására a valószínűségi termfrekvencia inverz dokumentumfrekvencia (PrTFIDF) módszert használtunk, 4,000 hosszú termfrekvencia vektorral.

4.5. A felhasználómodellre és a linkkiemelő rendszerre tett megszorítások

A felhasználómodell minden elérhető oldalt kiértékelt a saját osztályozói és súlyvektora alapján. Mindig azok közül az oldalak közül választott, melyeket még nem látogatott meg, és ezek közül a legmagasabb értékűre lépett tovább.

4.5.1. Felhasználómodell a mesterséges modellen

A felhasználót modellező súlyvektor két nem-nulla komponenst tartalmazott, melyek $+1$ vagy -1 értéket vehettek fel. Ez a súlyvektor ugyanahhoz a véletlenszerűen generált osztályozóvektorhoz tartozott, mint a linkkiemelő súlyvektora.

4.5.2. Felhasználómodell és linkkiemelő a letöltött adatbázison

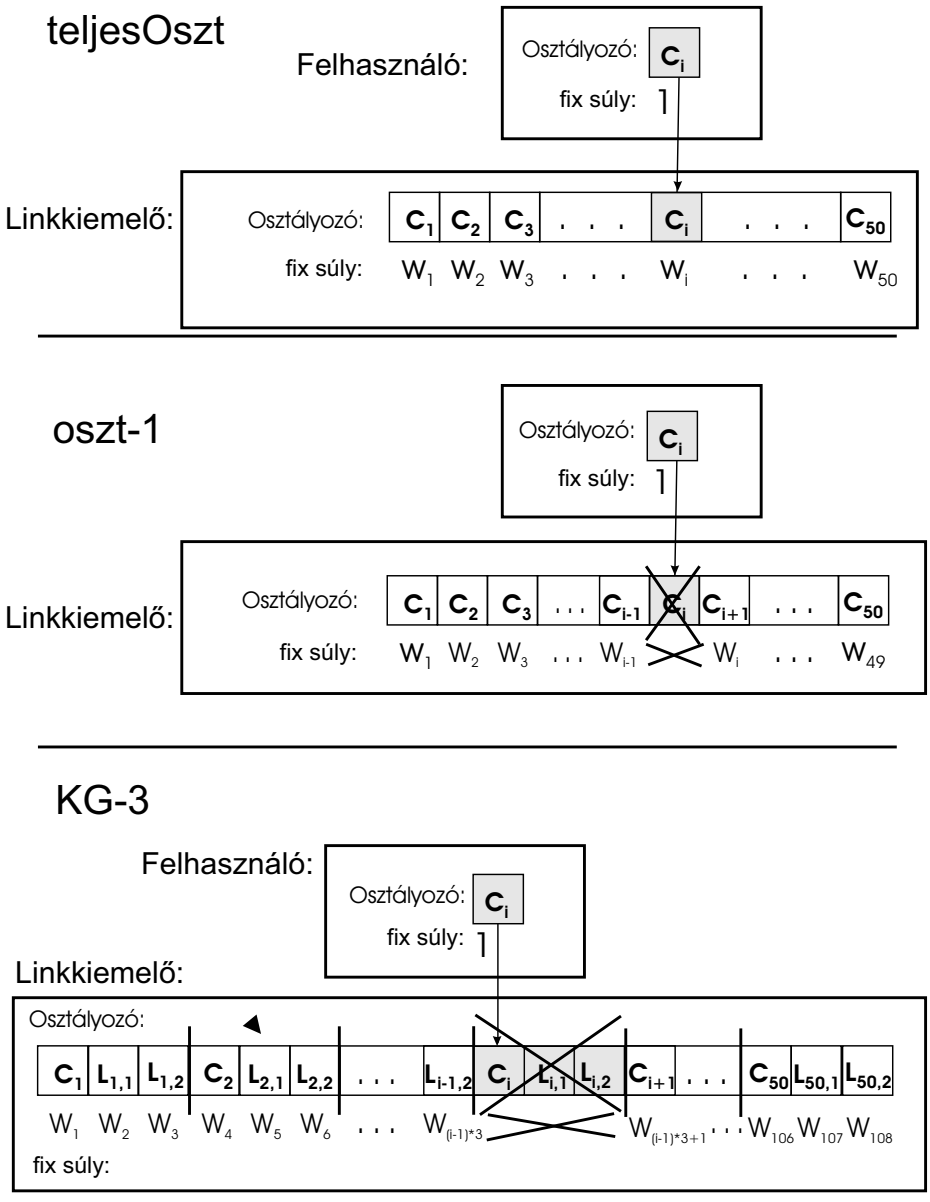
A felhasználót a Boley osztályok segítségével modelleztük. Két osztály jellemzett egy felhasználót, $+1$ vagy -1 súlyokkal, a mesterséges modell esetéhez hasonlóan. A linkkiemelő mind az 50 Boley osztályhoz tartozó osztályozót használhatta, tehát elég volt a felhasználómodell súlyvektorát megbecsülnie.

4.5.3. Felhasználómodell és linkkiemelő az Interneten

Három különböző típusú tesztet végeztünk az Interneten, az ezekben szereplő a felhasználómodell és linkkiemelő rendszer szerkezetét a 5. ábrán vázoltuk fel.

Az első típusú tesztekben (továbbiakban: *teljesOszt séma*) a felhasználót egy Boley osztályozó definiálta, 1-es súllyal. A linkkiemelő rendszer tartalmazta az összes Boley osztályozót, ugyanúgy, mint a letöltött adatbázison végzett tesztek esetében.

A második típusú tesztekben (továbbiakban: *Oszt-1 séma*) ugyanolyan felhasználómodellt használtunk, mint a *teljesOszt séma* esetében. A linkkiemelő rendszer

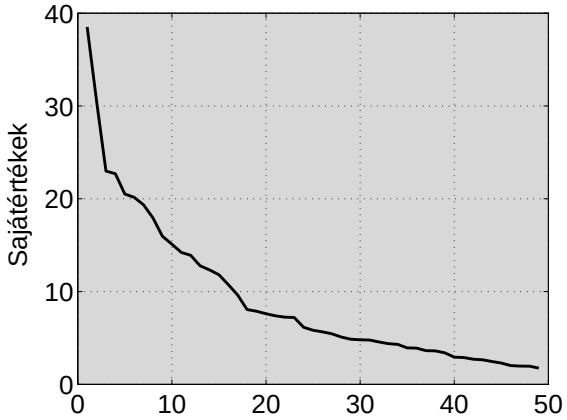


5. ábra. **Kísérleti sémák.**

A felhasználót mindegyik esetben az egyik Boley osztályozó határozta meg. Fent (teljesOszt séma): a linkkiemelő mindegyik Boley osztályt (C_i) tartalmazza. Középen: (oszt-1 séma) a linkkieemelőben a Boley osztályozók vannak, kivéve a felhasználómodell osztályozóját. Lent (KG-3 séma): minden Boley osztály és a hozzá tartozó környezeti gráf szintjeinek osztályozói (L_i) is résztvesznek a linkkiemelés-

viszont csak a felhasználót modellező osztályozón kívüli maradék 49 Boley osztályozót használhatta a tanulási folyamata során. Tehát ebben az esetben (az előzőekkel ellentétben) a felhasználómodell nem tartozott linkkiemelő osztályozóinak lineáris burkába, ezért a rendszernek más osztályok kombinációjaként kellett közelítenie a felhasználó célját. Mind a két sémában mind az 50 Boley osztályozóhoz tartozó felhasználómodellel végeztünk futtatásokat.

A PrTFIDF osztályozók termfrekvencia vektorának hossza 4000 dimenziós. Szinguláris értékfelbontással ellenőriztük, hogy az 50 frekvenciavektor nem lineáris kombinációja egymásnak (6. ábra). Tehát a felhasználómodell tökéletes azonosítása az *oszt-1 sémában* nem lehetséges, a kísérletek így tényleg olyan esetek ellenőrzését végzik el, amikor a felhasználó érdeklődése pontosan nem fejezhető ki a linkkiemelő rendszer osztályozóival.



6. ábra. **Az 50 Boley osztályozó szófrekvencia vektorai által alkotott mátrix szinguláris érték felbontása**

A mátrix mérete 50×4000, rangja 50. Ha az egyik osztályozót eltávolítjuk, a többi lineáris kombinációjából nem tudjuk előállítani.

A harmadik típusú tesztekben, melyet *KG-3 sémának* neveztünk el, ugyanazt a felhasználómodellt használtuk, mint az előző sémákban. A linkkiemelő rendszert viszont új típusú osztályozókkal bővítettük ki. A Boley osztályok köré kétszintű környezeti gráfokat próbáltunk építeni. A Boley osztály középpontjai voltak a megfelelő környezeti gráf középpontjai. A célunk az volt, hogy a környezeti gráf első és második szintjére is legalább 100 dokumentumot találjunk, ez 37 esetben sikerült (a környezeti gráf építés pszeudokódja megtalálható a függelékben). Ilyen módon 37×2 új szövegosztályt kaptunk, melyekre szintén betanítottunk PrTFIDF osztályozókat. A környezeti gráfokhoz - a középpont Boley osztályokat is beleszámítva - tehát $37 \times 3 = 111$ osztályozó tartozik, ezek alkották a linkkiemelő osztályozórendszerét. A felhasználómodellhez tartozó Boley osztályozót, és a köré épített környezeti gráf két szintjének osztályozóit viszont eltávolítottuk a linkkiemelőből, így valójában csak 108 osztályozóval dolgozhatott. Így ez a séma is a nehezebb típusú feladatot valósította meg. A tesztek során mind a 37 lehetséges felhasználómodellel végeztünk futtatásokat.

4.6. Adaptációs algoritmusok

A következőekben áttekintjük, hogy a linkkiemelő súlyvektorának frissítésére milyen adaptációs algoritmusokat használtunk.

4.6.1. Megerősítéses tanulás

A megerősítéses tanulás algoritmusa lineáris súlyokra:

$$\delta(t+1) \;=\; r(t+1) \;+\; \gamma\, \mathbf{W}^T\, \mathbf{S}\,(t+1) \;-\; \mathbf{W}^T\, \mathbf{S}(t) \tag{12}$$

$$\mathbf{W}(t+1) \;=\; \mathbf{W}(t) \;+\; \alpha\, \delta \mathbf{S}(t+1), \tag{13}$$

ahol

$$V(S(t)) \;=\; \mathbf{W}^T(t)\,\mathbf{S}(t)$$

$V(s) \in \mathbb{R}$ a dokumentumok értékbecslése, $\mathbf{W}(t) = (W_1(t), \ldots, W_n(t))$ a linkkiemelő súlyvektora a t-edik lépésben, $\mathbf{S}(t) = (S_1(t), \ldots, S_n(t))$ az osztályozók kimeneti vektora a felhasználó által a t-edik lépésben meglátogatott dokumentumra, γ a diszkont faktor, $r(t+1)$ az azonnali jutalom a felhasználó $t+1$-edik lépése után, $\delta(t+1)$ az értékbecslés hibája, α a tanulási ráta, amely függhet az időtől és (13) a súlyvektor frissítési szabálya. Az azonnali jutalom nulla volt, ha $\mathbf{W}^T(t)\,\mathbf{S}(t+1)$ a legnagyobb értékű volt az összes elérhető dokumentum közül, különben -1. Különböző diszkont faktorokat is kipróbáltunk, köztük a diszkontálás nélküli esetet ($\gamma = 1.0$) is. A megerősítéses tanulás bizonyos feltételek mellett garantáltan konvergál az optimális megoldáshoz. Ezen feltételek között szerepel, hogy az állapotok mind ismertek és markoviak, hogy a jutalom lehet sztochasztikus, de az eloszlásának állandónak kell lennie. Ezek a feltételek nem teljesülnek az internetes barangolás esetén.

4.6.2. Csúszóablakos technika

A második módszer, a csúszóablakos technika, tanulási szabálya a következő:

$$\mathbf{W}(t+1) \;=\; (1-\alpha)\mathbf{W}(t) \;+\; \alpha\mathbf{S}(t+1)$$

Itt az α tanulási ráta $1/(\kappa * t)$, ahol κ-t 0.1-nek választottuk. Ez a szabály a $J = \frac{1}{2}||\mathbf{W} - \mathbf{S}||^2$ négyzetes hibát minimalizálja a Robins-Monroe kritérium szerint. A tanulási módszer hibája, hogy a súlyok akkor is adaptálódnak a bemenethez, ha már jól modellezik a felhasználót. Mivel a súlyvektort a felhasználó által választott oldalak osztályozóvektorához hangoljuk, ezért implicit feltételezzük, hogy a felhasználó mindig jó oldalakat választ. Ez viszont nem teljesül akkor, ha olyan környezetbe tévedünk, ahol egyáltalán nincsen jó elérhető oldal. Ilyen esetben a már jól beállított

súlyok is elállítódhatnak.

4.6.3. Értékbecslési hibával modulált csúszóablakos technika

A harmadik módszer kiküszöböli a tanulási ráta megválasztásának problémáját. A szabály a tanulási rátát minden lépésben az értékbecslés sikerességének függvényében határozza meg.

$$\delta(t+1) \;=\; \mathbf{W}^T(t)\,(\mathbf{S}^*(t+1) \;-\; \mathbf{S}(t+1)) \tag{14}$$

$$\mathbf{W}(t+1) \;=\; \frac{(1-\alpha\delta(t+1))\mathbf{W}(t)+\alpha\delta(t+1)\mathbf{S}(t+1)}{|(1-\alpha\delta(t+1))\mathbf{W}(t)+\alpha\delta(t+1)\mathbf{S}(t+1)|}, \tag{15}$$

ahol $\mathbf{S}(t+1)$ az osztályozók kimeneti vektora a felhasználó által választott legjobb dokumentumra, $\mathbf{S}(t+1)$ pedig az értékbecslés alapján legjobb dokumentumra. α-t konstansnak választottuk, hogy a tanulási képesség a futás során ne csökkenjen. $\delta(t+1)$ a $t+1$. lépésben számolt értékbecslési hiba. Könnyen látható, hogy a felhasználó által választott oldal értékbecslése a 15. szabály alkalmazásával nőni fog, ha az S vektort minden lépésben normalizáljuk, tehát teljesül, hogy $|\mathbf{S}(t)| = 1$ minden $t = 1, 2, ...$-re. Ebben az esetben $\mathbf{W}(t+1)^T\mathbf{S}(t+1) \geq \mathbf{W}(t)^T\mathbf{S}(t+1)$.

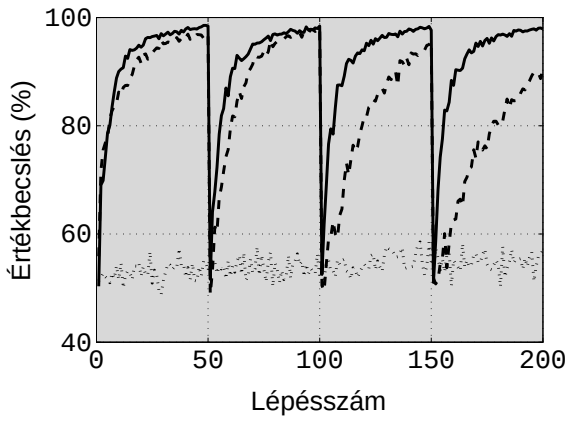
Ha a súlyvektor megfelelően modellezi a felhasználó értékelését, akkor abban az esetben sem fognak elállítódni a súlyok, ha olyan környezetbe tévedünk, ahol a felhasználó nem tud jó oldalt választani. A súlyok nagy módosításához ugyanis az is szükséges, hogy az értékbecslési hiba nagy legyen, ami jól beállított súlyok esetén nem lehetséges.

Bár a módszer értékbecslésre épül, az állapotok értékbecslését nem hosszútávra összegezve határozza meg, ezért nem tekinthető igazi megerősítéses tanulásos módszernek. Közelebbi kapcsolatban áll az értékelőfüggvény approximációját végző módszerekkel, mint például a STAGE [6].

5. Eredmények

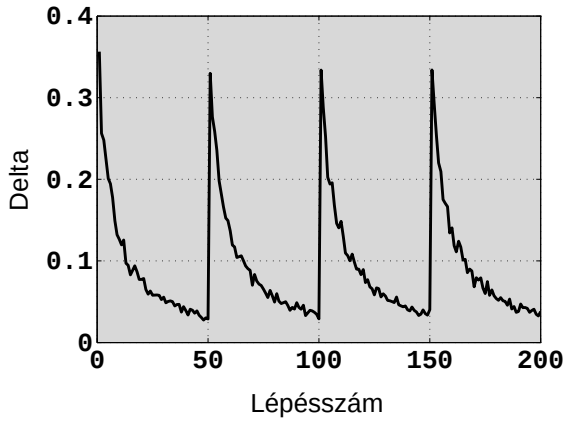
5.1. Eredmények a mesterséges modellen

A mesterséges modellen végzett futtatások célja az adaptációs algoritmusok összehasonlítása volt. Az eredmények 230 független futtatás átlagaként adódtak. Minden futtatás 200 lépésen keresztül tartott, az úkbóli adaptáció sebességének ellenőrzésére a felhasználómodellt 50 lépésenként véletlenszerűen megváltoztattuk. Az értékbecslés hatékonyságát láthatjuk a 7. ábrán.



7. ábra. **Adaptációs módszerek összehasonlítása**.
Az ábra az értékbecslés hatékonységét mutatja, 230 futtatás átlagában. Folytonos vonal: értékbecslési hibával modulált csúszóablakos technika, szaggatott vonal: csúszóablakos technika (tanulási ráta: $1/t$), pontozott vonal: megerősítéses tanulás. A felhasználómodell 50 lépésenként véletlenszerűen megváltozott.

A megerősítéses tanulás (pontozott vonal) igen gyengén teljesített, lényegében nem volt képes tanulásra a futás során. (Fontos megjegyezni, hogy az 50%-os értékbecslés a véletlen választásnak felel meg.)



8. ábra. **Értékbecslési hiba az idő függvényében**.
A felhasználómodell súlyvektora 50 lépésenként változott meg, látható hogy az értékbecslési hiba hirtelen növekedése jelzi a célváltozások bekövetkezésének időpontját.

A csúszóablakos technika (szaggatott vonal) elég jól teljesített, viszont a csökkenő tanulási ráta fokozatosan rontotta a tanulási képességét. Az optimális tanulási ráta megválasztása igen nehéz. Sőt, minden körülmények között jó tanulási ráta nem is létezik, hiszen ha kis tanulási rátát választunk, akkor lassú lesz az adaptáció, ha nagyot, akkor a rendszer nem képes finomhangolásra.

Az értékbecslési hibával modulált csúszóablakos technika konstruktív módon oldja meg a tanulási ráta választásának problémáját: a tanulási ráta nagysága attól függ, hogy éppen akkor mekkora a rendszer értékbecslésének hibája. Látható, hogy ezzel a módszerrel igen jó eredményeket kapunk, minden célváltozás után gyorsan lezajlik az újbóli adaptáció.

Az értékbecslési hiba mértékének figyelése egy új alkalmazási lehetőséget is ad. Amikor az értékbecslési hiba nagyon megugrik, akkor a felhasználó valószínűleg barangolás közben megváltoztatta célját, és új téma után kezdett érdeklődni (8.

ábra).

5.2. Eredmények a letöltött adatbázison

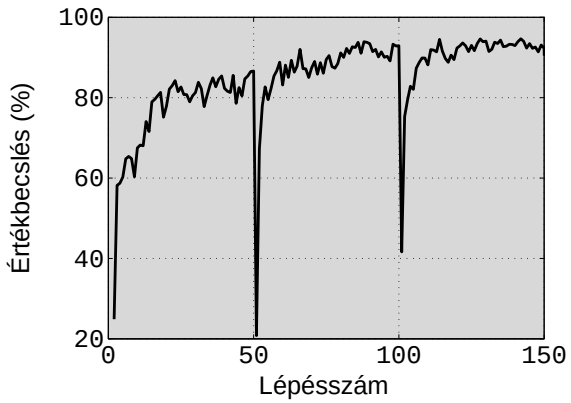
Értékbecslési hibával modulált csúszóablakos technikával a letöltött Geocities adatbázison végzett futtatásaink átlagos eredménye a 9. ábrán látható. A felhasználómodell ebben az esetben is 50 lépésenként célt változtatott. A viszonylag gyenge eredmények könnyen javíthatóak akkor, ha azokat az oldalakat is elérhetőeknek tekintjük a felhasználó számára, amelyeket már meglátogatott. Enélkül az adatbázis alacsony átlagos oldalankénti linkszáma (kb. három) rontja a teljesítményt. Ugyanis ha egyáltalán nincsen jó elérhető oldal, akkor a felhasználó nem képes céljának megfelelő linket választani, ezért – jó példák hiányában – a rendszer nem képes közelíteni a felhasználói szándékot. Ha mindez párosul azzal, hogy a véletlenszerűen választott kezdőpont érdektelen a felhasználó számára, akkor általában csak lassan tud eljutni el olyan környezetbe, ahol választásai kifejezik az őt érdeklő témát. Így természetesen az értékbecslés lassan fog javulni. A későbbi lépések során az egyre több elérhető oldal csökkenti az adatbázis alacsony átlagos linkszámának káros hatását, és láthatóan javul a linkkiemelő értékbecslése.

A linkkiemelő tehát az oldalankénti alacsony linkszám esetén erősen támaszkodik az „előzmények"-ben található hivatkozásokra. Tekinthető úgy is, hogy egyfajta kényelmes, szűrt elérést biztosít az előzményekben található nagyszámú oldalhoz.

5.3. Eredmények az Interneten

Négy különböző vizsgálatot folytattunk az Interneten, mindegyik esetben az értékbecslési hibával modulált csúszóablakos technikát használtuk.

Az első vizsgálatban a *teljesOszt séma* szerint, a másodikban az *oszt-1 séma* szerinti futtatásokat végeztünk. Mindkét esetben kimerítő tesztek zajlottak le, mind



9. ábra. **Eredmények a Geocities adatbázison.**

A felhasználót két 1 súlyú osztályozó modellezte, a linkkiemelő az összes Boley osztályozót használta. Minden korábban látott linket minden lépésben elérhetőnek tekintettünk („vissza" gomb, „előzmények"). A felhasználómodell célja 50 lépésenként véletlenszerűen megváltozott. A súlyokat az értékbecsléshibával modulált csúszóablakos technikával frissítettük.

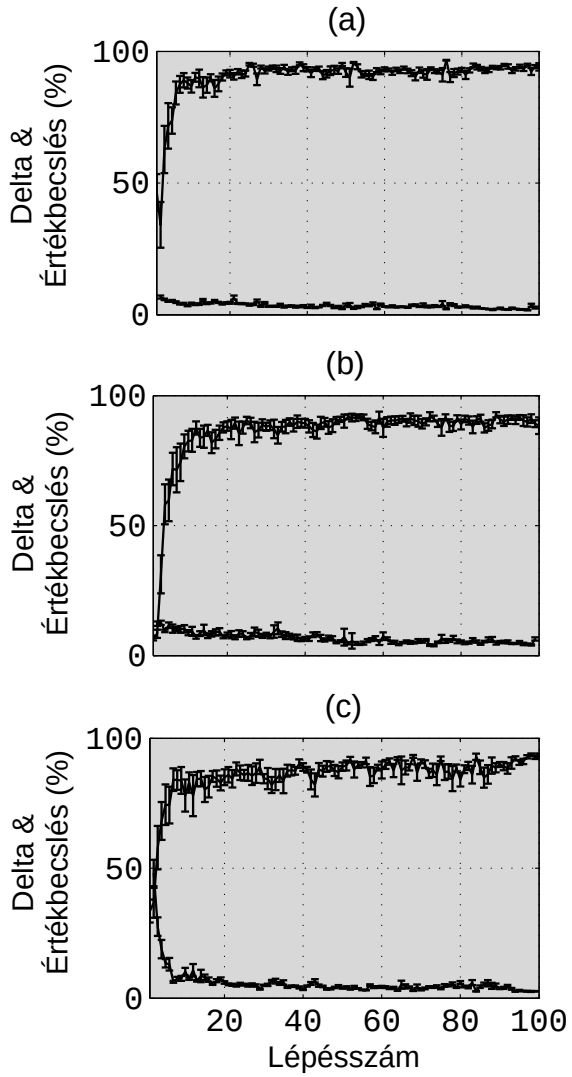
az 50 Boley osztályozóhoz tartozó felhasználómodellt kipróbáltuk egy-egy 100 lépéses futtatás során.

A harmadik vizsgálatban a *KG-3* séma szerinti futtatásokat végeztünk. Ebben a vizsgálatban is teljes tesztet hajtottunk végre, mind a 37 olyan osztályozó, amely köré sikerült környezeti gráfot építeni, szerepelt felhasználómodellként.

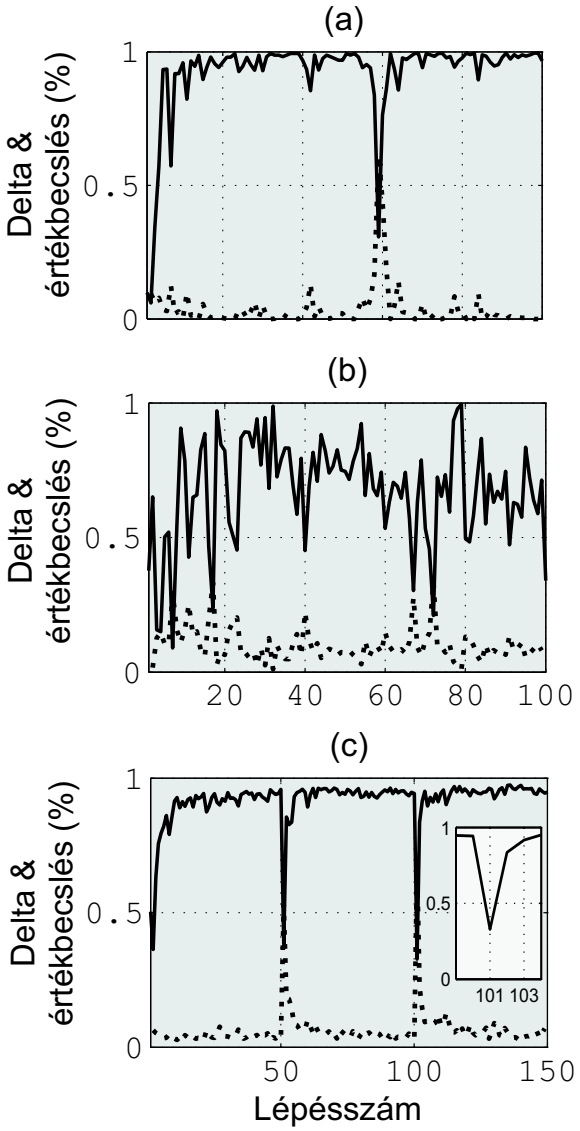
Az első három vizsgálat eredményei a 10. ábrán láthatók. Az ábrákon varianciákat ábrázoltunk, mert ha szórásokat rajzoltunk volna az átlag köré, akkor időnként 100% fölé kerültek volna a hibasávok. Tehát az átlagos teljesítmény jó volt, de időről-időre, felhasználómodellről-felhasználómodellre nagy szórást mutatott. Ezért a 11. (a) és (b) ábrán bemutattunk egy-egy –szubjektív megítélés szerinti – legjobb

és legrosszabb esetet a második vizsgálat eredményeiből.

A negyedik vizsgálatban a *teljesOszt séma* szerint futtattunk olyan 150 lépéses teszteket, melyek során a felhasználómodellt 50 lépésenként megváltoztattuk. A linkkiemelő osztályozóiból ekkor végig kimaradt az a három osztályozó, amely a futtatás során megjelenő felhasználómodellek valamelyikéhez tartozott (11. (c) ábra). Látható, hogy az adaptáció kezdetben viszonylag lassú, de a célváltozások során már nagyon felgyorsul, 2 lépésre is lecsökken az újbóli adaptáció ideje. Ez annak köszönhető, hogy kezdetben nem állnak rendelkezésre igazán jó oldalak, később viszont – az előzmények nagy száma miatt – az elegendő elérhető link közül rögtön lehet megfelelőeket választani.



10. ábra. **Eredmények az Interneten.** (a): *teljesOszt séma*, (b): *oszt-1 séma* (c): *KG-3 séma*. Fenti görbék: értékbecslés hatékonysága, lenti görbék: értékbecslési hiba. A görbék az összes lehetséges felhasználómodellre vett futtatás ((a) és (b): 50 db, (c): 37 db) átlagát ábrázoljuk, a varianciák feltüntetésével.



11. ábra. **Példák egyedi esetekre.** (a): A legjobb internetes futási eredmény *oszt-1 sémában.* (b): A legrosszabb internetes futási eredmény *oszt-1 sémában.* (c): Változó célú felhasználómodelles eredmény *teljesOSzt sémában.* Belső ábra: az adaptáció sebessége a 100. lépésnél bekövetkező célváltozás után.

6. Értékelés

6.1. Adaptációs módszerek értékelése

A csúszóablakos technika viszonylag jó eredményt adott, de érzékeny a tanulási rátára. Az értékbecslési hibával modulált csúszóablakos technika viszont minden lépésben előállít egy megfelelő tanulási rátát, megoldva ezzel a paraméterválasztás problémáját. Ha az értékbecslési hiba nagy, tehát a súlyok nem jól becsülik a felhasználói szándékot, akkor a tanulási ráta is nagy értéket vesz fel. Ha viszont a hiba kicsi, tehát a súlyok már nagyjából beálltak a megfelelő szintre, akkor a tanulási ráta is kicsi lesz, így a súlyok sem fognak erősen módosulni. Másrészt viszont a 7. ábrán látható, hogy a futtatás elején a sima csúszóablakos technika jobban teljesített, tehát az értékbecslési hiba segítségével előállított tanulási ráta nem optimális. Ennek ellenére a módszer jónak tekinthető, hiszen a futás egészében optimális tanulási ráta nem is létezik, mert – mint már említettük – az alacsony tanulási ráta nem teszi lehetővé a gyors tanulást, a magas tanulási ráta pedig megakadályozza a már nagyjából jól beállított súlyok finomhangolását.

A megerősítéses tanulás az értékbecslés hibáját csak azoknak az oldalaknak a tulajdonságai alapján becsüli, ahová a felhasználó korábban már lépett. Nem használja azoknak az oldalaknak a tulajdonságait, amelyeket ő ajánlott a felhasználónak, de az mégsem lépett oda. Ez egybevág a megerősítéses tanulás azon tulajdonságával, hogy tapasztalt tulajdonságok alapján becsüli az állapotok értékelésének hibáját:

$$\delta(t+1) \;=\; r(t+1) \;+\; V(\mathbf{S}_{tapasztalt}(t+1)) \;-\; V(\mathbf{S}_{tapasztalt}(t)) \qquad (16)$$

ahol $\mathbf{S}_{tapasztalt}(t)$ a tapasztalt állapot a t időpillanatban, $V(.)$ az értékfüggvény és $r(t)$ az azonnali jutalom. A tapasztalt állapot a mi esetünkben tehát annak a dokumentumnak a reprezentációja (osztályozóvektora), amelyre a felhasználó éppen

lépett.

Az értékbecslési hibával modulált csúszóablakos technika viszont a következőképpen számolja az értékbecslés hibáját:

$$\delta(t) \; = \; V(\mathbf{S}_{tapasztalt}(t+1)) \; - \; V(\mathbf{S}_{vrt}(t+1)) \tag{17}$$

ahol a „várt" az értékbecslés alapján legjobb állapotot jelenti. Érdekes, hogy ezt az információt használva a mi problémánkon sokkal gyorsabb tanulási szabályt kapunk.

Viszont ez a szabály is interpretálható megerősítéses tanulás algoritmusként, ha megfelelően defináljuk az azonnali jutalmat. Legyen a jutalom a következő:

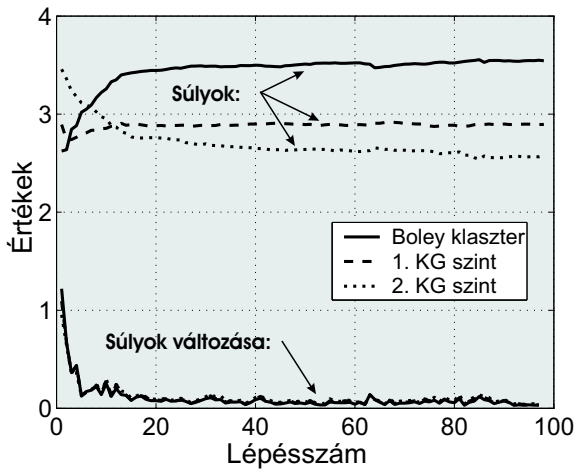
$$r(t+1) \; = \; V(\mathbf{S}_{tapasztalt}(t)) \; - \; V(\mathbf{S}_{vrt}(t+1)) \tag{18}$$

Ha ezt behelyettesítjük a (12) egyenletbe, akkor az a (17) egyenlettel azonossá válik. Tehát módszerünk megerősítéses tanulásos módszerként is bemutatható lenne, de a „várt" állapottal kapcsolatos információkat a megerősítéses tanulásos módszerek nem szokták ismertnek tekinteni, csak ha tervezés vagy jövőbeli állapotok kiértékelése is be van építve a rendszerbe.

6.2. Az osztályozók jelentősége

A három internetes futtatási séma során az eredmények hasonlóan jók lettek. Kicsi különbséggel a *teljesOSzt séma* teljesített a legjobban, a *KG-3 séma* pedig a legrosszabbul, de a különbségek hibahatáron belül vannak.

Nyilván a *teljesOszt séma* a legegyszerűbb feladat, hiszen ott a felhasználómodell célja explicit szerepel a linkkiemelő osztályozói között, csak meg kell találni. Az *oszt-1 sémában* végzett futtatások hasonlóan jó eredményei azt jelzik, hogy az osztályozók kifejezőereje együtt meghaladja a hozzájuk tartozó osztályok külön-külön vett felismerését. Ez alapján remélni lehet, hogy sok felhasználói cél kifejezhető 50 db egyszerű, általános Boley osztályozóval.



12. ábra. **Súlyok és változásuk.**
A súlyok abszolútértékei (felső görbe) és a súlyváltozások abszolútértékei (alsó görbe) a *KG-3* séma 3 különböző típusú osztályozói szerint összegezve.

Ha minden Boley osztály köré sikerült volna környezeti gráfot építeni, akkor biztosan mondhatnánk, hogy a *KG-3 séma* linkkiemelője nagyobb kifejezőerővel rendelkezik, mint az előző esetek rendszerei, mert Boley osztályozókon kívül más osztályozókat is használhat a felhasználó lépéseinek predikciójához. Így viszont csak annyit állíthatunk, hogy több osztályozóval rendelkezett, és hogy ezek között sok más módon meghatározott szövegosztályra betanított osztályozó is volt. Mint láttuk, a környezeti gráfok használata előnyösnek bizonyult témaspecifikus keresések esetén. Ennek ellenére nem sikerült a korábbiaknál jobb eredményeket felmutatni a tesztek során. Ennek oka lehet, hogy az osztályozók nagyobb száma miatt a linkkiemelő súlyvektora is hosszabb, így a tanulási feladat nagyobb keresési térben zajlik. A 12. ábrán látható, hogy a környezeti gráfokhoz tartozó súlyok ugyanannyit változtak a futtatás során, mint a Boley osztályokhoz tartozók.

Az eredmények alapján az 50 Boley osztályozó használata ésszerű választásnak tűnik.

Az, hogy milyen témákban tud hatékonyan segíteni a linkkiemelő, erősen függ az osztályozók által lefedett témáktól. Nem várhatjuk például, hogy egy erősen speciális orvosi témában való keresésnél a pontos cél jellegét képes legyen az 50 általános osztályozónk leírni. A linkkiemelő osztályozói természeten bármikor kicserélhetők, az algoritmus osztályozófüggetlen. Ezért megtehetjük, hogy bizonyos témákban specializált osztályozórendszereket készítünk. Így lehet például egy sebészeti témákban jártas linkkiemelőnk, mely jól ki tudja fejezni a sebészeten belüli konkrétabb témák közötti különbségeket, más tárgyban viszont nem alkalmas keresésre. Így elkerülhetjük azt, hogy túlontúl általános, mégis mindenben szakértő linkkiemelőt készítve, túl sok osztályozót helyezzünk a rendszerbe, és ezzel – a magas dimenziószám miatt – megnehezítjük a tanulási feladatot. Egy teljesen személyre szabott rendszer tehát több szinten képes a felhasználó tudásához, érdeklődéséhez alkalmazkodni. Egyrészt lehet több, a számunkra érdekes témákra specializált osztályozórendszerünk, ezek közül minden böngészéshez azt választhatjuk ki, amelyik tárgyban éppen keresni kívánunk. Ezután az osztályozókban tárolt tudást felhasználva, a gyorsan adaptálódó súlyvektor fogja – az általánosabb témán belül – a konkrét célunkat kifejezni. Az osztályozókat betaníthatjuk mi is, vagy használhatunk mások által kifejlesztett és közzétett rendszereket.

További kutatási téma lehet a Boley osztályok közötti különbségek, az osztályok jellegzetességeinek feltárása. Megfigyeltük, hogy erősen témafüggő a linkkiemelő teljesítménye. Valószínűleg bizonyos témák nem tartoztak az osztályok lineráis együttese által kifejezhető témák közé.

Az osztályozók kiválasztására esetleg automatikus rendszert is ki lehetne dolgozni: a linkkiemelő néhány kiindulási osztályozót használ, majd az alacsony súlyt

kapottakat kidobja, a magasabbakat megtartja. Illetve a magas súlyt kapott osztályozót helyettesítheti az ő témájának résztémáira betanított, speciálisabb osztályozókkal. Ehhez egy hierarchikus, fastruktúrájú osztályozórendszerre lenne szükségünk, melyek közül a linkkiemelő választaná ki az éppen használatba kerülőket.

Nem szabad figyelmen kívül hagynunk, hogy egy emberi felhasználó böngészése a modellfelhasználóénál sokkal zajosabb információkat nyújt a linkkiemelő számára. Sokszor ráklikkelünk egy linkre, majd rögtön látjuk, hogy a mutatott oldal nem érdekes számunkra. Ezeket a dokumentumokat kár lenne a linkkiemelőnek pozitív példának tekintenie, ezért szükséges valamilyen eljárás a felhasználó téves döntéseinek kiszűrésére. Egyik lehetőség az, hogy figyeljük a felhasználó további viselkedését, például az oldalakon töltött idő mennyiségét. A másik lehetőség, hogy direkt megerősítést kérünk a felhasználótól (mint például [19]-ban), például minden releváns oldal megtalálásakor elvárjuk, hogy nyomjon meg egy gombot, és csak a gomb megnyomásakor történt lépéseket tekintjük sikereseknek.

A módszer igazi alkalmazhatóságát természetesen majd a valódi emberi felhasználóval végzett tesztek eredményei dönthetik el.

6.3. Linkkiemelés, monitorozás és barangolás közti különbségek

A *linkkiemelés* segíti a böngészésünk közben érintett területek közvetlen környezetének gyors áttekintését, ezzel rövidítve a keresésre fordított időt. A linkkiemelő gyors adaptációja kulcsfontosságú, ugyanis a felhasználó gyorsan türelmét veszítheti. A keresett információ helye ismeretlen, csak reméljük, hogy a közelébe kerülünk, és könnyen lehet, hogy el is megyünk mellette.

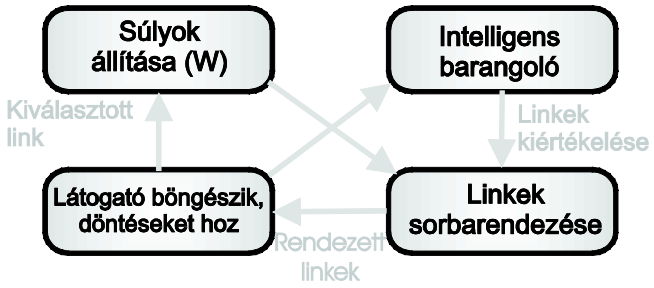
A *monitorozás* esete (pl. GoogleNews) ettől több szempontból is különbözik. A keresett információ helye ismert, hiszen a monitorozás csak konkrét oldalak frissítését

figyeli. Ezeken az oldalakon nem is veszíthet el érdekes információt. A monitorozás tehát egy egyszerűbb probléma teljeskörűbb megoldása. Nem igényel adaptációt, viszont biztos, hogy nem találja meg az új információkat a monitorozott oldalakon kívüli helyeken, és jelentős sávszélességet igényel a folyamatos használata.

Az intelligens *barangoló* több szempontból is átmenetinek tekinthető a linkkiemelés és a monitorozás között. A keresett információ helye ismeretlen, viszont felhasználói közbeavatkozás nélkül is nagyobb területet járhat be. A témaspecifikus működéshez elengedhetetlen az adaptáció, ugyanis az Internet különböző régiói különböző értékbecslést igényelnek, ahogy ez az irodalomban bemutatásra került [19, 15].

A linkkiemelő, barangoló és monitorozó rendszerek a klasszikus keresőkkel és adatbázisokkal egymást kiegészítve szolgálják a felhasználót az információ gyors megtalálásában. Mindegyikükre szükség lehet attól függően, hogy mi a keresésünk célja. A linkkiemelő, barangoló és monitorozó rendszerek a friss információk kutatásakor lehetnek igazán hasznosak, hiszen ez a nagy keresőrendszerek gyenge pontja.

A linkkiemelő és barangoló egy lehetséges együttműködési sémáját mutatja a 13. ábra. Az intelligens barangoló a felhasználó távollétében őt érdeklő oldalak után kutathat. Amikor a felhasználó leül böngészni, ellenőrizheti a barangoló által ajánlott oldalakat, és pozitív illetve negatív megerősítést kapcsolhat hozzájuk. Ezután a tényleg releváns oldalakról indulva elkezdhet böngészni a linkkiemelő további segítségével. Egy lehetséges taktika, hogy bízzuk a barangolóra a már jól ismert témák kutatását, és a linkkiemelő segítségével mi végezzük a felfedezőbb jellegű keresést új témák után, új helyeken böngészve. Ilyen exploratív böngészés esetén gyakori, hogy gyakran váltogatjuk célunkat, ezekben az esetekben a linkkiemelőnek gyorsan reagálnia kell az oldalak megváltozott értékelésére.



13. ábra. **A felhasználó, az intelligens barangoló és a linkkiemelő együttműködésének sémája**

6.4. Egyéb alkalmazás

A módszer szerveroldalon is használható lehet, például dinamikus portálok létrehozásakor. A portál üzemeltetői a különböző felhasználói attitűdökhöz speciális tartalmat és személyreszabott hirdetéseket rendelhetnek. A böngészés közben hozott döntéseik alapján az ismertetett eljárással következtetni lehet a látogatók érdeklődésére. Ennek alapján pedig mód nyílik olyan linkek felajánlására, illetve olyan hirdetések megjelenítésére, amelyek nagy valószínűséggel érdeklik őket. Így a látogatók hamarabb megtalálják az őket érdeklő információt, elégedettebbek lesznek a portállal és legközelebb is ezt a helyet fogják választani. Ugyanakkor kevesebb fölösleges oldalt töltenek le, ezáltal értékes sávszélesség szabadul fel, amin több látogatót tud kiszolgálni a portál. Mint látható ez a technika a látogatóknak és a portál fenntartójának is előnyére válhat.

7. Köszönetnyilvánítás

Köszönettel tartozom témavezetőmnek, Dr. Lőrincz Andrásnak, aki lehetővé tette, hogy a Neurális Információfeldolgozási Csoport keretein belül a tudományos kutatómunkát megismerhessem. A három év alatt, amíg itt dolgoztam, mindvégig folyamatos segítséget kaptam tőle, akár a munkám végzéséről, akár szakmai fejlődésemről volt szó. A nemzetközi szintű csoport – mely egyedülálló lehetőséget nyújt a diákok számára a tudományos munkába való bekapcsolódásra – vezetése mellett személyes felelősségének érzi az egyes diákok pályájának alakulását is.

Köszönettel tartozom továbbá Palotai Zsoltnak, aki szinte második témavezetőként kísérte végig munkámat, idejét és energiáját nem kímélve bármikor segített nekem.

Köszönöm Mandusitz Sándornak, hogy bármikor fordulhattam hozzá kérdéseimmel, és a csoport összes tagjának, hogy velük dolgozhattam.

8. Függelék

8.1. A környezeti gráf építésének pszeudokódja

BUILDING CONTEXT GRAPH

PSEUDO-CODE

1. *Initialization*

   root docs = documents in the cluster

   new empty context graph (CG)

   branching ratio $\leftarrow n$ ($n = 2.5$)

   min number of root docs := m ($m = 100$)

   0th level of CG := 0th level of CG + root docs

   sort root docs by its distances from the cluster center

2. *Constructing the Context Graph*

   for the first 200 root docs:

      if CG IS READY **then**

         remove documents from the 0th level of CG which doesn't linked by any

            document in the 1st level of CG

         CG is ready, STOP

      page := next doc in root docs

      SEARCH FOR BACKWARD LINKS TO LEVEL 0(page)

      for all new d documents in the 1st level of CG:

         SEARCH FOR BACKWARD LINKS TO LEVEL 1(d)

3. *Checking the size of Context Graph*

   if number of documents in each level of the CG > 100 **then**

      remove documents from the 0th level of CG which doesn't linked by any

         document in the 1st level of CG

      CG is ready, STOP

   else Building of CG failed

end BUILDING CONTEXT GRAPH

function CG IS READY

```
    return (size of the 0th level of CG >= min number of root docs and
            size of the 1st level of CG >= (size of the 0th level of CG) * branching ratio and
            size of the 2nd level of CG >= (size of the 0th level of CG) * (branching ratio)^2)
end function

proc Search For Backward Links(level, doc)
    for each l link in doc:
        if level=0 and there is a link on l to doc then
            Add To Context Graph(l)
        if level=1 and
              there is a link on l to any document in the 1st level of CG then
            Add To Context Graph(l)
end proc

proc Add To Context Graph(doc)
    for each level i 1 to 2:
        if there is a link in doc to a document at the (i-1)th level and
        the number of documents pointing to that document < 2*branching ratio and
        the average number of documents pointing to a document < branching ratio and
        the CG doesn't contain doc at smaller levels then
            add doc to the ith level of CG
end proc
```

Hivatkozások

[1] R. Albert and A.-L. Barabási, *Statistical mechanics of complex networks*, arXiv:cond/0106096v1, June 2001, `http://www.nd.edu/ alb`.

[2] Christopher Bailey, Samhaa R. El-Beltagy, and Wendy Hall, *Link augmentation: A context-based approach to support adaptive hypermedia*, OHS-7/SC-3/AH-3, 2001, pp. 239–251.

[3] Marko Balabanovic, *An adaptive web page recommendation service*, First Internation Conference on Autonomous Agents, 2000, pp. 378–385.

[4] A. L. Barabási, R. Albert, and H. Jeong, *Scale-free characteristics of random networks: The topology of the world wide web*, Physica A **281** (2000), 69–77.

[5] D.L. Boley, *Principal direction division partitioning*, Data Mining and Knowledge Discovery **2** (1998), 325–244.

[6] J. A. Boyan and A. W. Moore, *Learning evaluation functions to improve optimization by local search*, Journal of Machine Learning Research **1** (2000), 77–122.

[7] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener, *Graph structure in the web*, Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking, North-Holland Publishing Co., 2000, pp. 309–320.

[8] S. Chakrabarti, D.A. Gibson, and K.S. McCurley, *Surfing the Web backwards*, 8th World Wide Web Conference (Toronto, Canada), 1999, `http://www8.org/w8-papers/5b-hypertext-media/surfing/`.

[9] S. Chakrabarti, M. van der Berg, and B. Dom, *Focused crawling: a new approach to topic-specific Web resource discovery*, Computer Networks (Amsterdam, Netherlands: 1999) **31** (1999), no. 11–16, 1623–1640.

[10] Liren Chen and Katia Sycara, *WebMate: A personal agent for browsing and searching*, Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98) (New York) (Katia P. Sycara and Michael Wooldridge, eds.), ACM Press, 9–13, 1998, pp. 132–139.

[11] Thomas M. Cover and Joy A. Thomas, *Elements of information theory*, Wiley Series in Telecommunications, John Wiley & Sons, New York, NY, USA, 1991.

[12] M. Diligenti, F. Coetzee, S. Lawrence, C. Lee Giles, and M. Gori, *Focused crawling using context graphs*, 26th International Conference on Very Large Databases, VLDB 2000 (Cairo, Egypt), 10–14 September 2000, `http://www.neci.nec.com/ lawrence/papers/ focus-vldb00/ focus-vldb00.ps.gz`.

[13] Thorsten Joachims, *A probabilistic analysis of the rocchio algorithm with tfidf for text categorization*, Proceedings of the Fourteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., 1997, pp. 143–151.

[14] Thorsten Joachims, Dayne Freitag, and Tom M. Mitchell, *Web watcher: A tour guide for the world wide web*, IJCAI (1), 1997, pp. 770–777.

[15] I. Kókai and A. Lőrincz, *Fast adapting value estimation based hybrid architecture for searching the world-wide web*, Applied Soft Computing **2** (2002), 11–23.

[16] R.K. Kolluri, N. Mittal, R.P. Ventakachalam, and N. Widjaja, *Focused crawling*, 2000, `http://www.cs.utexas.edu/ users/ramki/ datamining/ fcrawl.ps.gz`.

[17] S. Lawrence, *Context in web search*, IEEE Data Engineering Bulletin **23** (2000), 25–32.

[18] H. Liebermann, *Letizia: an agent that assists web browsing*, Proceedings of the 14th International Joint Conference on Artificial Intelligence, 1995, pp. 924–929.

[19] A. Lőrincz, I. Kókai, and A. Meretei, *Intelligent high-performance crawlers used to reveal topic-specific structure of the WWW*, Int. J. of Founds. of Comp. Sci. (2002), no. 13, 477–495.

[20] A. McCallum, K. Nigam, J. Rennie, and K. Seymore, *Building domain-specific search engines with machine learning techniques*, AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace, 1999, `http://www.cs.cmu.edu/~mccallum/papers/cora-aaaiss98.ps`.

[21] Filippo Menczer, W. Nick Street, Narayan Vishwakarma, Alvaro E. Monge, and Markus Jakobsson, *Intellishopper: a proactive, personal, private shopping assistant*, Proceedings of the first international joint conference on Autonomous agents and multiagent systems, ACM Press, 2002, pp. 1001–1008.

[22] S. Mukherjea, *WTMS: A system for collecting and analyzing topic-specific web information*, 9th World Wide Web Conference, 2000, `http://www9.org/w9cdrom/293/293.html`.

[23] J. Rennie, K. Nigam, and A. McCallum, *Using reinforcement learning to spider the web efficiently*, Proceedings of ICML-99, 16th International Conference on Machine Learning (Bled, SL) (Ivan Bratko and Saso Dzeroski, eds.), Morgan Kaufmann Publishers, San Francisco, US, 1999, pp. 335–343.

[24] I. Schwab and W. Pohl, *Learning user profiles from positive examples*, 1999.

[25] R. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT Press, Cambridge, 1998.

[26] Stephan ten Hagen, Maarten van Someren, and Vera Hollink, *Exploration/exploitation in adaptive recommender systems*, Proceedings of the European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems (Oulu, Finland), 2003.