Eötvös Loránd Tudományegyetem
Informatikai Kar
Programozáselmélet és Szoftvertechnológiai Tanszék

Portálkészítés segítése gráf alapú módszerekkel

Diplomamunka

Pintér Balázs
Programtervező matematikus szak
Nappali tagozat

Témavezető:
dr. habil. Lőrincz András
Tudományos főmunkatárs

Budapest, 2010

Tartalomjegyzék

1. fejezet

Bevezetés

Napjainkban a világháló a legnagyobb rendelkezésünkre álló adathalmaz, ami rohamos ütemben bővül. A Google 2008. július 25-én jelentette be hivatalos blogján[1], hogy elérték az ezer milliárdadik egyedi webcímet és naponta több milliárd újat fedeznek fel.

1.1. Keresők

Egy ekkora és ilyen ütemben fejlődő adathalmaz kezelése hatalmas kihívás, ugyanis a megfelelő információk megtalálása nehéz. A kereskedelmi, webfelülettel rendelkező hagyományos webes keresők (pl. Google[2], Yahoo[3]) csak korlátozott mértékben alkalmasak erre a feladatra. Funkcionalitásban és teljesítményben is elmaradnak a kívánatostól.

A funkcionalitás fő hiányossága, hogy nem képesek új, fontos információkat önállóan javasolni a felhasználó számára, csupán a felhasználó által már valamennyire ismert információt tudják kiegészíteni. Meg tudjuk válaszolni segítségükkel, hogy mikor érkezett meg a Spirit marsjáró a Marsra, de ha

[1] `http://googleblog.blogspot.com`
[2] http://google.com
[3] `http://yahoo.com`

véletlenül marslakókat fedezne fel, arról nem értesülnénk a keresőmotorok használatával.

A technikai hiányosságuk a világháló nagyságából és nagy ütemű növekedéséből következik. A webnek csak egy részét képesek bejárni, továbbá nem képesek lépést tartani a változásokkal. Az egyik, nyilvánvaló következménye ennek, hogy amit nem térképezett fel a kereső által használt keresőrobot, azt nem találhatjuk meg a keresővel. A másik pedig az, hogy, mivel a keresőgép nem képes megfelelően követni a megváltozott weboldalakat, ezért információink nem lesznek naprakészek, hiszen a weblap egy előző változatából származnak.

Amikor a felhasználó elindítja a keresést, sokszor már több nap, hét, esetleg hónap is eltelt azóta, hogy a kereső szoftvere a keresett weboldalt meglátogatta. A kereső indexében, ami alapján kiértékeli a keresést és visszaadja a találatokat, a weblapok régi változatának reprezentációja található. Így a felhasználó nem talál meg olyan weboldalakat, amik ugyan tartalmazzák a kívánt információt, de a kereső látogatása után frissítették őket.

A dolgozat írásakor (2010. áprilisa és májusa), a Google kereső adatbázisában a `http://www.elte.hu/` webcím 1 napos, a `http://www.inf.elte.hu/` négy napos, az `http://www.inf.elte.hu/KARUNKROL/SZERVEZET/DEKANIHIVATAL/TO/Lapok/default.aspx`[4], egy hetes, a `http://www.inf.elte.hu/KARUNKROL/SZERVEZET/DEKANIHIVATAL/TO/SZIGORLAT_NAGYPROGRAM_ZAROVIZSGA/Lapok/ptmzvmenetrend.aspx`[5] három hetes késéssel volt indexelve. Egy példa több hónapos eltérésre a `http://progkor.inf.elte.hu/unix/magyar.man/index.html`: a dolgozat írásakor a 2010. február 2-ai verzió volt az adatbázisban. Lewandowski egy részletes, három

[4] a Tanulmányi Osztály honlapja

[5] a programtervező matematikus záróvizsga honlapja

évig[6] tartó vizsgálata szerint [21] a Google kereső adatbázisában egy weblap általában két napos késéssel volt indexelve.

A világháló nagysága egy új problémát is magában hordoz: a túlzott információbőség, más szóval *információ túlterhelés* problémáját. Még ha fel is tudnánk térképezni valós időben az egész webet, akkor is egyre nehezebb lenne megtalálni a kívánt információkat az egyre bővülő adathalmazban.

Az információ túlterhelés a felhasználó szempontjából is problémát jelent. Napjainkban annyi információ zúdul rá a világhálóról, hogy képtelen feldolgozni azt. Emiatt is lényeges az, hogy ki tudjuk válogatni a felhasználó számára érdekes, fontos információkat.

1.2. Portálok

Új és lényeges információk összegyűjtésére és közzé tételére a weben főleg a portálok[7] szolgálnak. A portálok több különböző helyről származó információkat tesznek elérhetővé egy webhelyen. E fő céljukon kívül egyéb funkciókkal, szolgáltatásokkal is rendelkezhetnek, mint például kommunikáció a felhasználók között[8].

A tematikus portálok azzal a céllal jöttek létre, hogy összefogják az egy témába tartozó legfontosabb webhelyeket. Hazánkban az egyik legjelentősebb tematikus portálgyűjtemény a `lap.hu` tartománynév aldoménjei (pl.: `elte.lap.hu`, `forma1.lap.hu`, `drakula.lap.hu`, `film.lap.hu`). Ezek a portálok a témában fontos webhelyek listáját, esetleg a listában szereplő webhelyekről gyűjtött érdekes híreket tartalmazzák.

A portálok előnye, hogy nem csak információk kiegészítésére alkalmasak,

[6] 2005-2007

[7] pl.: `startlap.hu`

[8] Ezen plusz funkciók nem tartoznak dolgozatom témájába, a továbbiakban a portálok csak mint információ szolgáltatók szerepelnek.

mint a keresők, hanem teljesen új információk közzé tételére is. Emellett megszűrik a felhasználóra zúduló információtömeget: csak a fontos, érdekes híreket teszik közzé.

Hátrányuk viszont, hogy a megfelelő információforrások kiválasztása nehéz. A portál készítőin múlik, hogy a felhasználók milyen információkat érnek el, illetve miről maradnak le. Nincs objektív mérce arra, hogy mennyire jók vagy rosszak a források. Továbbá a portálok információforrásainak listája ritkábban változik, mint egy keresőrobot adatbázisa, nem naprakész.

1.3. A keresőmotorok és a portálok ötvözése

Láttuk, hogy a portálok a világháló jó információ prezentáló eszközei lennének, ha meg lenne oldva az információforrások dinamikus és bizonyíthatóan megfelelő kiválasztása, ahol a megfelelő azt jelenti, hogy a felhasználó ne maradjon le a fontos hírekről. Dolgozatomban egy megoldást dolgozok ki erre a problémára, illetve a hírek közötti válogatás megkönnyítésére. A „megfelelő" fogalmát később matematikailag is definiálom.

A megoldás első lépése a portál témájának meghatározása. Annak, hogy előre meghatározzuk a portál témáját, két oka van. Egyrészt csökkentjük az információ túlterhelést mind a felhasználó, mind a rendszerünk felé. Másrészt a felhasználó egyszerre egy adott témával foglalkozik, egyik érdeklődési körébe tartozó információkat, híreket keres. Ez a tematikus portálok nagy számában is jelentkezik, illetve abban, hogy a nem tematikus portálok gyakran több tematikus portált összesítenek, ezekre hivatkozásokat tartalmaznak.

A megoldás bonyolultsága csökkent, mert a webnek csak az adott témához tartozó részével kell foglalkoznunk, nehézsége viszont nőtt, mert meg kell határoznunk ezt a részt.

A feladat tehát a webnek egy adott témával foglalkozó részének, és abban a legfontosabb híreket közlő, legnagyobb befolyással bíró webhelyek kiválasztása.

A megoldás két részből áll. Az első részben feltérképezzük a webnek az adott témával foglalkozó szegletét. Ehhez egy tematikus keresőrobotot készítettem, ami előállítja az általa bejárt terület egy gráf alapú modelljét. A gráfban a csúcsok a weblapok, a köztük futó élek pedig a hiperhivatkozások. Ezen a modellen információ kaszkádokat keresek, amik egy-egy hír terjedési részgráfjának felelnek meg. A cél a lehető legtöbb ilyen kaszkád detektálása lehetőleg minél kevesebb webhely figyelésével. Egy webhely akkor detektál egy kaszkádot, ha legalább a kaszkád egyik csúcsa a webhelyen található weblap, más szóval a webhelyen közlik a hírt.

A legbefolyásosabb webhelyek megkeresése kombinatorikusan nehéz feladat, mivel azt keressük, hogy *összességében* mennyi kaszkádot detektálnak a kiválasztott webhelyek, és egy kaszkádot több webhely is detektálhat. Erre egy szubmoduláris mohó gráf alapú algoritmust használok, melynek hatékonysága matematikai tételekkel igazolható. Annak eldöntésére, hogy egy weboldal egy adott témába tartozik-e, a természetes nyelvfeldolgozás módszereit és támogató vektor gépeket[9] használtam.

Végeredményként megkapjuk a legnagyobb befolyással rendelkező webhelyek listáját. Mivel kevés, 20-40 webhely szükséges ahhoz, hogy a hírek nagy részét, 80-90 százalékát detektáljuk, ezek a felhasználó számára kezelhetők, egy tematikus portál hivatkozáslistájába beilleszthetőek.

Ahhoz, hogy a legfontosabb híreket megtaláljuk, két eszköz áll rendelkezésünkre. Az egyik a hírhez tartozó információ kaszkád nagysága. Minél nagyobb egy kaszkád, azaz minél több webhely közli, annál fontosabb. Vi-

[9]angolul: support vector machine

szont ez még nem elegendő, jó lenne, ha lenne egy hatékony eszköz arra, hogy egy hír témáját, tartalmát ránézésre meg lehessen állapítani. Erre a célra egy kulcsszó kivonatoló rendszert fejlesztettem, ami képes dokumentumok kulcsszavakkal való jellemzésére.

Ezekkel az eszközökkel a portálkészítés folyamatának nagy része automatizálható. A keresőrobot fel tudja deríteni és le tudja tölteni az adott témával foglalkozó webhelyeket és weblapokat. A szubmoduláris algoritmus meg tudja határozni a legfontosabb, legnagyobb befolyással bíró webhelyeket. Az információ kaszkádok nagyságából lehet következtetni az egyes hírek fontosságára. A kulcsszókivonatoló algoritmus segítségével pedig a hírek százai, ezrei áttekinthetővé válnak.

2. fejezet

Tematikus anyagok gyűjtése a világhálón

2.1. Bevezetés

A hagyományos keresőrobotok a weboldalakat a világháló linkstruktúrájának bejárásával gyűjtik (2.1. ábra). Egy előre megadott *kezdeti webcím halmazból* indulnak, és folyamatosan új, az eddig letöltött weblapok kimenő hiperlinkjeiről elérhető weboldalakat találnak és töltenek le. A webet egy gráfnak, a weblapokat csúcsoknak, a hiperhivatkozásokat pedig éleknek felfogva a keresőgép ezt a gráfot járja be valamilyen keresési stratégia szerint.

A már megtalált, de még nem letöltött weblapok címei a *nyílt csúcsok halmazában*[1] tárolódnak. A keresés kezdetén a nyílt csúcsok megegyeznek a kezdeti webcím halmaz elemeivel. A keresőgép minden lépésben választ egy webcímet a nyílt csúcsok halmazából, letölti azt a weblapot, amire a webcím mutat és kinyeri belőle a hiperhivatkozásokat. A hivatkozott, de még nem meglátogatott webcímeket hozzáadja a nyílt csúcsok halmazához.

[1] angolul: crawl frontier

A feldolgozása után a webcím kikerül a nyílt csúcsok halmazából.

Azt már a bevezetőben is láthattuk, hogy a világháló hatalmas mérete és fejlődési üteme miatt egy keresőrobot sem képes azt teljes egészében felfedezni. Emiatt a keresőgépek fontossági sorrendet állítanak fel a letöltendő weblapok között, különböző keresési stratégiák[2] szerint választják a következő feldolgozandó webcímet a nyílt csúcsok halmazából. Például az olyan jól ismert keresési stratégiák, mint a szélességi keresés alkalmazhatóak. Egy másik stratégia a PageRank stratégia, ami az webcímeket a Google PageRank algoritmusa szerint rendezi sorba.

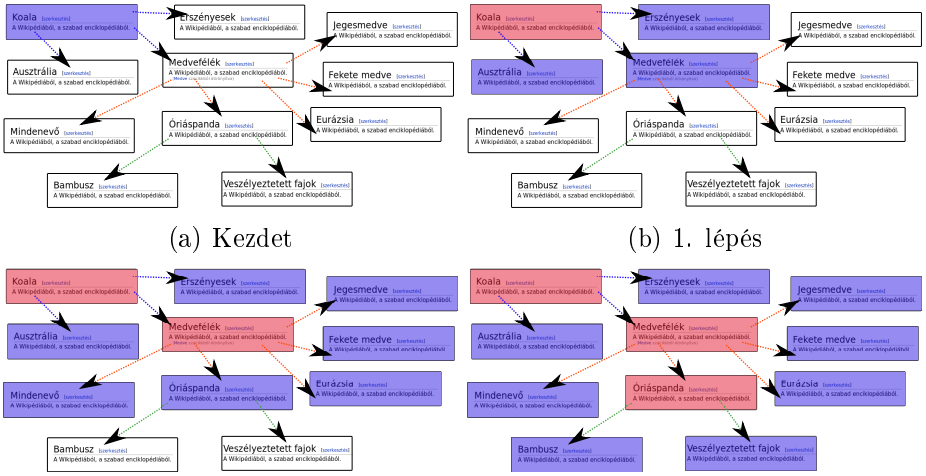
A tematikus keresőrobotok (2.2. ábra) csak egy előre meghatározott témába tartozó weboldalakat töltenek le. Azt, hogy egy weblap az adott témába tartozik-e, még azelőtt kell eldönteni, mielőtt letöltöttük. Ezért a weboldal tartalmára nem támaszkodhatunk, csak a letöltése előtt is meglévő információk állnak rendelkezésünkre.

Az egyik első tematikus keresővel, a *Fish Search*-csel [8] kezdődően ezt a problémát többféleképpen közelítették meg. A közös ezekben a megközelítésekben, hogy mindegyik az eddig bejárt weboldalak jellemzőit használva dolgozik, esetleg háttértudást is igénybe véve, és egy igaz/hamis értéket (az oldal a témába tartozik-e vagy sem), vagy egy pontszámot (mennyire tartozik az oldal a témába) állítanak elő. A legtöbb tematikus keresőrobot ezután a legjobbat először stratégia alapján tölti le a *nyílt csúcsok halmazában* szereplő webcímeket.

A *Fish Search*-ben úgy döntik el egy webcímről, hogy releváns weblapra mutat-e, hogy összehasonlítják kulcsszavakkal vagy reguláris kifejezésekkel. Ennek továbbfejlesztése, a *Shark Search* [13] a horgonyszövegek[3], a linket

[2]angolul: crawling policy

[3]angolul: anchor text, a hiperlink szövege, amire rá lehet kattintani



(a) Kezdet (b) 1. lépés

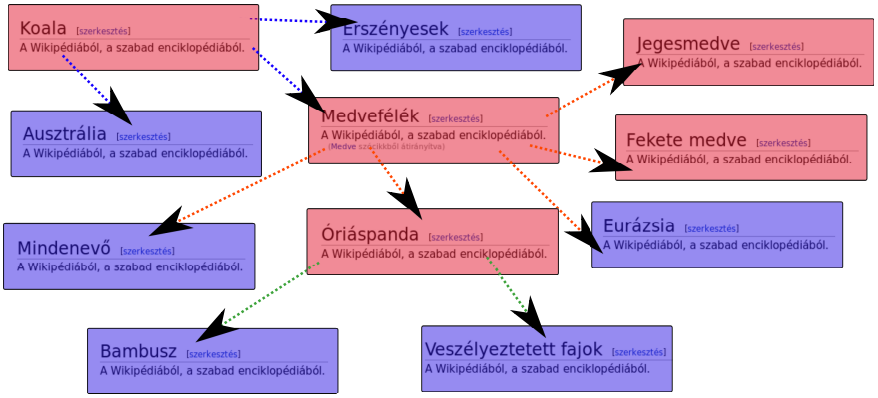
(c) 2. lépés (d) 3. lépés

2.1. ábra. **Egy keresőmotor működése.** Az ábrán egy keresőmotor működése látható a magyar Wikipedia egy kis részén. A kék csúcsok a nyílt halmazt, a pirosak a letöltött és feldolgozott weblapokat jelölik. Az élek a weblapokat összekötő hiperhivatkozások, színük a kezdeti webcím halmaztól való távolságukat jelöli. Kezdetben az egyetlen nyílt csúcs a „Koala", ezt az első lépésben letöltjük és feldolgozzuk. Azokat a weboldalakat, amikre hivatkozás mutat belőle, beletesszük a nyílt halmazba. A második lépésben a „Medvefélék" oldalt dolgozzuk fel, a nyílt halmaz az ebből kiinduló hivatkozások céljaival bővül. A harmadik lépésben az „Óriáspanda" weblap hivatkozásainak céljaival bővítjük a nyílt halmazt. Az egyes lépésekben letöltendő és feldolgozandó weblapot egy *keresési stratégia* szerint választjuk ki a nyílt halmazból.

körülvevő szövegrész, és a weblap őseinek[4] pontszáma alapján számolja ki a pontszámot. Cho és mások [5] a PageRank algoritmus alapján rendelték a relevancia értékeket a webcímekhez, de nem értek el javulást a szélességi kereséshez képest. Menczer szerint [25, 26] azért, mert a PageRank túl általános a tematikus feladatokhoz.

Pant és Srinivasan [33] a következőképpen osztályozzák a weblapok témába tartozásának eldöntéséhez felhasznált információkat:

[4]a weblaphoz vezető út csúcsai



2.2. ábra. **A tematikus kereső stratégia** Egy ideális tematikus kereső csak azokat a weblapokat tölti le, amik az előre meghatározott témával, esetünkben a medvefélékkel foglalkoznak.

- *A hivatkozás szövegkörnyezete*: A szülő oldalon, azaz a weboldalra linkelő oldalon, a rá mutató webcím körül található szöveg. Ez lehet az egész oldal, vagy csak egy része. Szinte az összes tematikus keresőrobot használja [1, 3, 8, 10, 13, 16, 24, 31, 36].

- *A weblap ősei*: A weblaphoz vezető úton található más weboldalakból kinyert információk, legtöbbször a szövegük. Diligenti és mások [10], illetve Johnson és mások [16] próbálkoztak a szülő oldalon felül a távolabbi ősök használatával, azonban Johnson arra jutott, hogy a további ősök hozzávétele nem javít a tematikus keresőrobot teljesítményén.

- *Web gráf*: A világháló gráfjának az a része, ami a weblapot körülveszi. Például, ha a szülő oldal „jó” elosztó[5], akkor weblap pontszámát növeljük. Ezt a stratégiát először Chakrabarti és mások [3] alkalmazták, azonban nem mutatták meg a hasznosságát. Pant és Menczer [32] statisztikailag szignifikáns javulást talált a stratégia alkalmazásakor.

[5] angolul: hub

Többen is alkalmaztak tanulóalgoritmusokat egy weboldal relevanciájának meghatározására. Az első osztályozót is használó tematikus keresőrobot Chakrabarti és mások [3] nevéhez fűződik. A kereső egy taxonómiát használ, amin a felhasználó bejelölheti az őt érdeklő témákat. Egy naiv Bayes osztályozó számolja ki annak a $P(t|o)$ valószínűségét, hogy az o oldal a t témába tartozik. Diligenti és mások [10] kontextus alapú tematikus keresőgépet alkottak. Egy letöltött weblap és a releváns weblapok közötti hivatkozásszámot becslik Naiv Bayes osztályozókkal. Johnson [16] támogató vektor géppel dönti el, hogy egy weblap a témába tartozik-e. Az osztályozókon kívül egyéb tanulóalgoritmusokat, például megerősítéses tanulást [22,23,30,35], vagy genetikus algoritmusokat [4] is alkalmaztak már a probléma megoldásában.

Ebben a fejezetben egy új típusú, a webhelyek szerint szerveződő adaptív tematikus keresőrobotot hozok létre. A keresőrobot alapja az a feltételezés, hogy az egy webhelyen található weblapok azonos témáról szólnak. A tematikus keresés feladatának egy dekompozícióját állítom elő, ahol az egyes webhelyeken különálló keresések folynak, amik lehetnek tematikusak is. Minden lépésben kiválasztunk egy webhelyet a rajta eddig talált releváns weboldalak relatív gyakorisága alapján (ez a tematikus stratégiánk), amiről letöltünk egy weblapot. A keresőrobot hosszú távon is jól teljesít: 4,1 millió letöltött dokumentum között a releváns weboldalak aránya magas, 70% feletti akkor is, ha a weblapot nem tematikus kereső stratégiával választjuk ki.

A bevezetés további részében bemutatom a keresőgép szervező elvét, a *webhely lokalitás* fogalmát, majd a keresőgép működését tekintem át. A fejezet következő három alfejezete a keresőgép, az adaptáció vagy folyamatvezérlés, és a weblapok relevanciájának eldöntésére használt osztályozó részletes bemutatását tartalmazza.

2.1.1. A téma lokalitás fogalma és kiterjesztése

A tematikus keresőrobotok a téma lokalitás elvén[6] [7] működnek q(2.3. ábra). E szerint a legtöbb hiperhivatkozás hasonló témájú weboldalakat kapcsol össze. Így, ha egy kereső eddig egy adott témába tartozó weblapokat töltött le, akkor a nyílt csúcsok halmazának elemei között is sok weblap van ebből a témából.

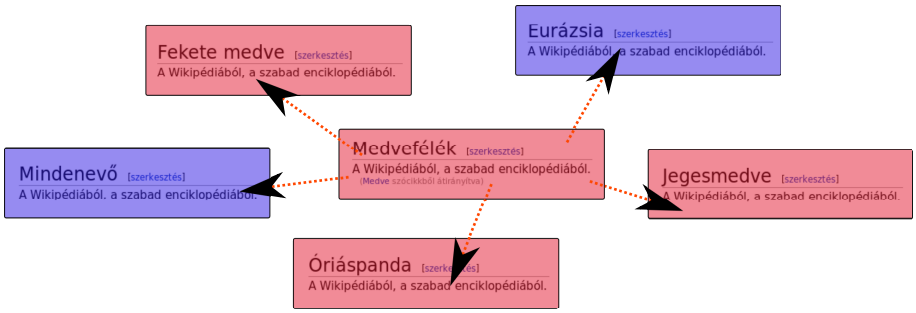
A fejezetben kiterjesztem a téma lokalitás fogalmát, és egy új típusú, webhely alapú keresőgépet készítek. A kereső jóságát és hatékonyságát nagyléptékű kísérletekben igazolom, összesen több, mint négymillió weboldal letöltésével. Kísérleteim során a *gyűjtési ráta*[7] folyamatosan magas maradt, a futtatások végén a releváns (témába tartozó) weblapok aránya meghaladta a 70 százalékot. A keresőrobot hatékony, párhuzamosított architektúrával rendelkezik. Folyamatosan alkalmazkodik a környezetéhez, fenntartva a magas gyűjtési rátát és sebességet.

A téma lokalitás fogalmát a következőképpen terjesztem ki. Nem csak azt feltételezem, hogy egy hiperlink hasonló témájú oldalakat köt össze, hanem azt is, hogy az egy webhelyen található weboldalak hasonló témáról szólnak. Másként fogalmazva, ha egy webhelyről választunk véletlenszerűen két weblapot, azok jobban összefüggnek, mint ha az egész világhálóról választanánk kettőt. Később látni fogjuk, hogy a kiterjesztés egyszerűsége ellenére egy szervezési elvet ad a keresőrobot építéséhez, és nagy hatékonyságú gyűjtést tesz lehetővé. A továbbiakban ezt a fogalmat *webhely lokalitásnak* hívom.

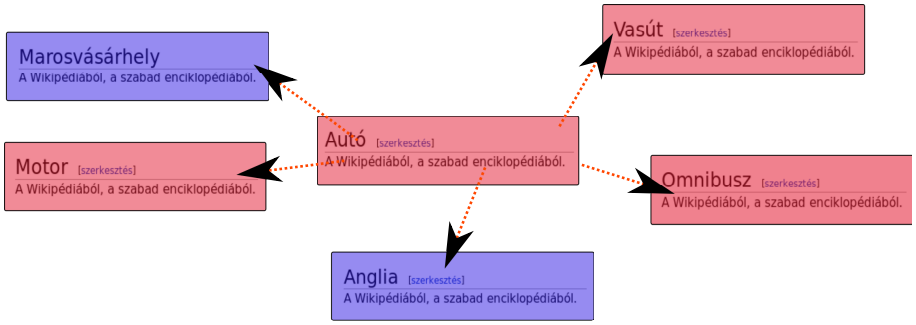
A webhely lokalitás lehetőséget ad arra, hogy egy új, tágabb kontextust, az egész webhelyet figyelembe véve szervezzük meg a keresési stratégiát (2.4. ábra). A tematikus keresők a weblapok szintjén dolgoznak, a téma loka-

[6] angolul: topical locality

[7] angolul: harvest rate, a barangolás egy pontjából visszafele tekintve egy adott időintervallumon a releváns és összes letöltött weboldal számának hányadosa



(a) Medvefélék



(b) Közlekedési eszközök

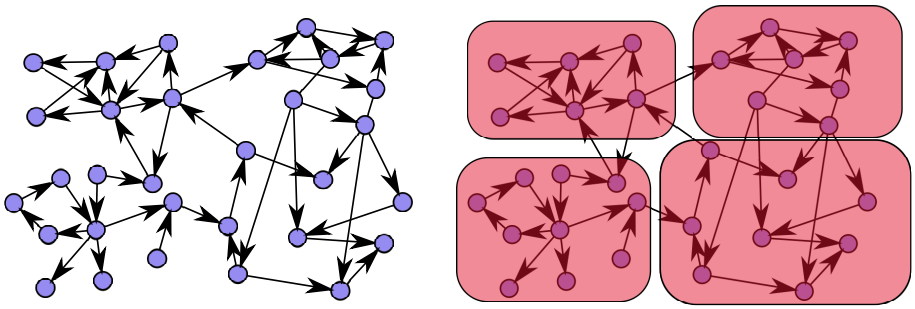
2.3. ábra. **A téma lokalitás elvének egy illusztrálása** Látható, hogy a magyar Wikipédián a „Medvefélék” weboldal sok medvefélérről szóló oldalra, míg az „Autó” sok közlekedési eszközzel kapcsolatos oldalra mutató hivatkozást tartalmaz. A témába tartozó weblapokat pirossal jelöltem.

litás elvét kihasználva: az eddig letöltött weblapok jellemzői alapján határozzák meg, hogy a nyílt csúcsok halmazának mely elemeiről valószínűsíthető, hogy a keresett témába tartozik. Dolgozatomban bevezetek egy új, magasabb szintet, a *webhelyek szintjét*. Keresőrobotom az egyes webhelyek adott témába tartozásának valószínűségét becsli. Más szóval azt becsüljük, hogy az egyes webhelyekről letöltött következő weblap mekkora valószínűséggel lesz a témába tartozó.

Tehát azt, hogy melyik weblapot töltsük le a nyílt csúcsok halmazából, két lépésben döntjük el, a két különböző szinten (2.5. ábra). A felső, webhely szinten meghatározzuk a webhelyet, amiről letöltjük a weblapot, a minden

egyes webhelyhez tartozó *relevancia százalék* alapján. A *relevancia százalék* a webhely témába tartozásának a valószínűsége, amit a keresőgépben az eddig letöltött releváns oldalak relatív gyakoriságával becslünk. Itt tematikus kiválasztást alkalmazunk, azaz a magasabb relevancia százalékkal rendelkező webhelyekről gyakrabban töltünk le.

Második lépésként az alsó, weblap szinten kiválasztjuk a letöltendő weboldalt erről webhelyről. Az itt alkalmazott keresési stratégia lehet tematikus is. Ha tematikus stratégiát alkalmazunk, a rendszer több témáról szóló webhelyek esetén is jó hatásfokkal működhet. Mivel a stratégia a webhelynek a minket érdeklő témáról szóló részét fogja letölteni, a relevancia százalékokat jól fogjuk tudni becsülni a magasabb szinten. Dolgozatomban a webhely alapú kereső stratégia eredményességét vizsgálom, ezért a weboldalakra nem alkalmazok tematikus stratégiát. Az eredmények azt támasztják alá, hogy a kereső így is jól működik.



(a) Hagyományos keresőrobot

(b) Webhely lokalitást használó keresőrobot

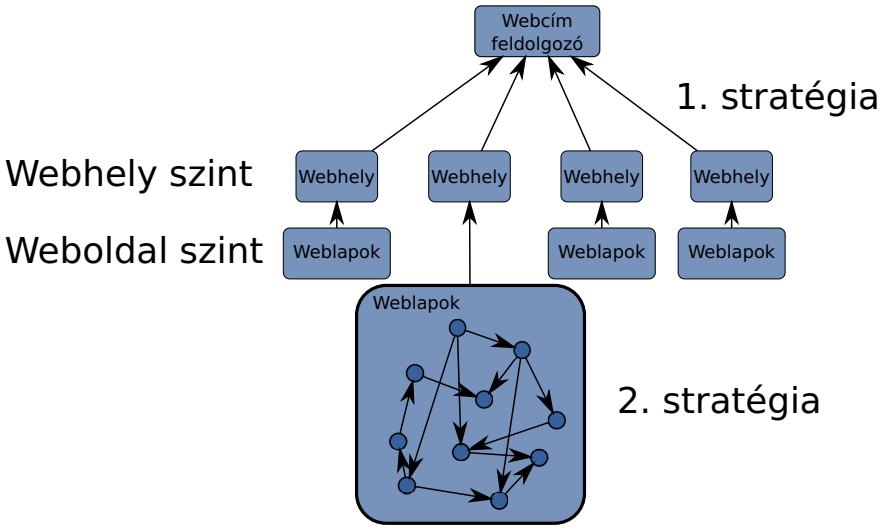
2.4. ábra. **A webhely lokalitást használó tematikus keresőrobot.** A webhely lokalitás fogalma szerint az egy webhelyen található weboldalak hasonló témáról szólnak. Egy hagyományos kereső a weblapok (kék pontok) közötti hiperhivatkozások alapján keres. A webhely alapú keresőrobot egy új, tágabb kontextust, az egész webhelyet (piros téglalapok) is figyelembe véve szervezi meg a keresési stratégiát. Így ha egy webhely az adott témáról szól, gyakrabban, ha pedig valószínűleg másról, akkor nagyon ritkán töltünk le róla weblapot.

Látható, hogy a vázolt rendszer a tematikus keresés problémájának egy dekompozíciója. Ez több előnnyel is jár. A keresőgép hatékonyan párhuzamosítható. A webhelyeken dolgozó weblap szintű lokális keresőgépeknek egymással nem kell kommunikálniuk. A kommunikáció csak a magasabb webhely szinten szükséges, viszont ott a kis számításigény miatt nincs is szükség párhuzamosításra. Szükséges még, hogy a weblap szintű keresőgépek elküldjék a magasabb szintre az éppen feldolgozott weblapról, hogy a témába tartozott-e, de ez csekély adatforgalmat eredményez.

Az általam megvalósított rendszer a párhuzamos architektúrájú Heritrix keresőgépre épül [27]. Mint látni fogjuk, a szerkezete kissé eltér az imént felvázolttól, például a már meglátogatott weblapok adatbázisa globális információ. Ez nyilvántartható lenne az egyes webhelyekhez külön-külön is, nem igényelne kommunikációt a weblap szintű modulok között. Az eredményeket ezek az implementációs kérdések nem befolyásolják.

A dekompozíció másik előnye, hogy csökkenti a feladat bonyolultságát. A keresőgép hatékonyabb lehet, mivel az adatszerkezetek mérete és az algoritmusok futási ideje is csökken. Ez azt jelenti, hogy a keresőrobot nagyobb léptékű barangolásokra is képes lehet.

Kísérleteimben a webhely lokalitáson alapuló keresőrobot eredményességét vizsgáltam. Emiatt csak a magas szinten használok tematikus kiválasztást: az egyes weblapok témába tartozását csak aggregáltan, a webhelyek *relevancia százalékaiban* veszem figyelembe, a weblap szintű keresőkben nem. Ennek ellenére az irodalomban közölt eredményekkel összehasonlítva is magas gyűjtési rátát értem el. A futtatások végén a releváns (témába tartozó) weblapok aránya meghaladta a 70 százalékot. Eredményeim a webhely lokalitás elv hasznosságát igazolják.



2.5. ábra. **A kétszintű keresőrobot** A keresőrobot két lépésben választja ki a következő letöltendő weblapot. Az 1. stratégia a webhelyet választja ki a relevancia százalékok alapján. A 2. stratégia ezután kiválaszt egy letöltendő weblapot a webhelyről A 2. stratégia hagyományos keresési stratégia, ami lehet tematikus is. Dolgozatomban a 2. stratégia nem tematikus, mert az 1. stratégia hatékonyságát vizsgálom.

2.1.2. Adaptáció, folyamatvezérlés

Ha a weben korlátozások nélkül barangolhatnánk, a gyűjtési rátát könnyen maximalizálni tudnánk. Egy lehetséges optimális algoritmus a webhelyeket relevancia százalék szerint sorbarendezné, és e szerint a sorrend szerint töltené le őket a lehető legnagyobb sebességgel.

Azonban, mivel több korláttal is szembe kell néznünk, ez az algoritmus a gyakorlatban kivitelezhetetlen. A témába tartozó honlapokat, webhelyeket nem ismerjük, csak a barangolás során fedezzük fel. Ahhoz, hogy a relevancia százalékokat meg tudjuk becsülni, le kell tölteni az adott webhelyről több weblapot is attól függetlenül, hogy ezek a témába tartoznak-e vagy sem. Egy másik korlát, hogy nem tölthetünk le az egyes webhelyekről olyan gyakran, mint ahogy szeretnénk, a nekik helyet adó szerverek túlterhelésének veszélye

miatt.

A legfontosabb gyakorlati probléma a tematikus keresőrobotnál az, hogy a barangolás folyamán az elérhető releváns weblapok száma ingadozik. Sokszor a webnek egy olyan területére tévedünk, ami nem tartalmaz a témába tartozó weboldalakat. Azon felül, hogy a gyűjtési ráta csökken, egy sokkal súlyosabb problémával is szembe kell néznünk. A téma lokalitás elve miatt a nem releváns weblapokról többnyire nem releváns weblapokra jutunk, emiatt ha egyszer eltévedt a barangoló, nehezen tud visszatalálni a webnek a megadott témával foglalkozó részére. Érdemesnek tűnik tehát a nem releváns honlapok letöltését megakadályozni adaptáció bevezetésével: a keresőrobot letöltési sebességét csökkenteni nem releváns honlapok esetén.

De tovább is vihetjük ezt a gondolatot: egy vezérlő elvvel megoldhatjuk mind az adaptációt, mind pedig a relevánsabb webhelyek gyakoribb mintavételezését. Ugyanis, ha a webhelyekről való letöltés gyakoriságát szabályozzuk, ezzel egyúttal szabályozni tudjuk a keresőrobot letöltési sebességét is. Minden webhelyhez tartozik egy *várakozási idő*, ami az adott webhelyről való két letöltés között minimálisan el kell, hogy teljen. A várakozási idő a relevancia függvénye. Ezen függvény segítségével valósítjuk meg az adaptációt: ha lecsökken a gyűjtési ráta, akkor leszűkítjük a barangolást a relevánsabb weboldalakra, ha pedig növekedni kezd, akkor kiterjesztjük.

2.1.3. A weblapok osztályozása

Egy tematikus keresőrobotnak el kell tudni dönteni, hogy a letöltött weblapok a számára érdekes témába tartoznak-e. Erre egy osztályozót készítettem. A feladat nehézségét növeli, hogy a világhálón szinte az összes lehetséges témában találhatóak dokumentumok. A negatív tanítópéldákat nehéz úgy összegyűjteni, hogy egy ilyen változatos dokumentumhalmazt lefedjenek.

Esetemben, mivel webhely alapú keresőgépet készítettem, az osztályozást meg lehet tenni az után is, hogy letöltöttük a weblapot. Az osztályozás eredménye nem arra kell, mint a hagyományos keresőrobotok esetében. Ott ugyanis azt kell eldönteni, hogy letöltsük-e a weblapot, míg esetemben a *relevancia százalékok* becslésére szolgál az osztályozó eredménye. Ez hatalmas előnyt jelent, mivel így a weboldalakat tartalom alapján tudom osztályozni, lehetővé téve a sokkal pontosabb osztályozást.

Az osztályozáshoz támogató vektor gépet [37] használok. A pozitív példákat az IMDB (Internet Movie Database) rec.arts.movies.reviews hírcsoport archívumából [8], a negatív példákat négy különböző, változatos témákat lefedő korpuszból gyűjtöttem. Így összesen 28 000 pozitív és 56 000 negatív tanítópéldával dolgoztam. Az osztályozó nagyon nagy pontossággal el tudja dönteni egy dokumentumról, hogy a témába tartozik-e: 10-szeres keresztvalidációval 0.995-ös F-mértéket kaptam.

2.2. A webhely alapú keresőrobot

Ebben a fejezetben a bevezetésben felvázolt keresőrobot felépítését, keresési stratégiáját és folyamatkezelését mutatom be részletesen. Mivel, mint ahogy a bevezetésben említettem, az alsó, weblap szinten hagyományos, nem tematikus keresési stratégiát használok, a keresőgép bemutatása a felső, webhely szintre szorítkozik.

2.2.1. A keresőrobot felépítése

A keresőrobot webhelyekkel dolgozik, egy webhelyet a tartománynevével azonosít. Nem teszünk különbséget az esetleg egy tartománynév alatt

[8] `http://www.imdb.com/Reviews/`

található, több különböző témáról szóló webhelyek között[9].

Az architektúra középpontjában az *aktív webhelyek listája* áll, ami az eddig felderített webhelyeket tartalmazza (2.6. ábra). A listában (*tartománynév*, *relevancia százalék*) párok vannak. A tartománynév azonosítja a webhelyet, a relevancia százalék pedig megmutatja, hogy mennyire releváns az adott webhely. Itt annak a valószínűségét becsüljük az előzőleg letöltött weboldalak alapján, hogy a következő weblap, amit a webhelyről letöltünk, releváns lesz:

$$relevancia\_szazalek_{webhely} = \frac{relevans_{webhely}}{mind_{webhely}} \tag{2.2.1}$$

, ahol a $relevans_{webhely}$ a webhelyről eddig letöltött releváns weblapok, a $mind_{webhely}$ pedig az összes, a webhelyről letöltött weblap száma. Azt, hogy egy weblap releváns-e, a tartalma alapján döntjük el egy támogató vektor gép segítségével[10].

A magasabb relevancia százalékkal rendelkező webhelyekről gyakrabban töltünk le weblapokat (2.7. ábra). Minden egyes webhelyhez külön *nyílt csúcsok halmaza* tartozik, ami a letöltésre váró weblapok webcímeit tartalmazza. Minden egyes webhelyhez tartozik egy *várakozási idő* is: ennyi időnek kell minimálisan eltelnie a webhelyről való két letöltés között. A várakozási idő a webhely relevancia százalékának és egy paraméternek a függvénye[11].

A keresőrobot indításakor az aktív webhelyek listáját egy kezdeti webhely listával inicializáljuk. Futás közben csak azokról a webhelyekről töltünk le weblapokat, amik rajta vannak a listán. A listát a téma lokalitás elve szerint folyamatosan bővítjük. Ha egy olyan webhellyel találkozunk, ami nincs benne

[9] részletesebben lásd alább

[10] lásd: 2.4. fejezet

[11] részletesebben lásd a 2.2.2. fejezetben

a listában, akkor egy másik listába, a *hivatkozott webhelyek listájába* tesszük. Ebben a listában azt tartjuk nyilván, hogy a benne lévő egyes webhelyekre az aktív webhelyek listájából hány hivatkozás történt. Ha a hivatkozások száma egy webhelynél meghalad egy előre beállított *hivatkozásszám küszöböt*, akkor a webhely átkerül az aktív webhelyek listájára, és elkezdjük a letöltését.

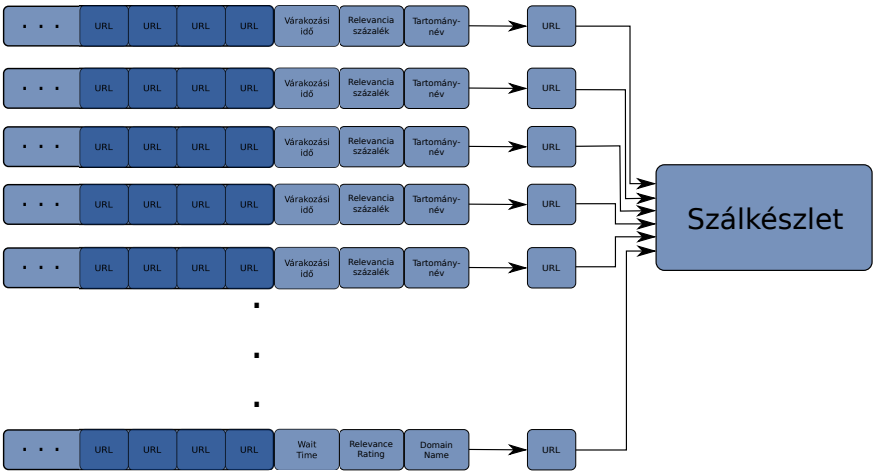
A keresőrobot implementációja a Heritrix nevű, nyílt forráskódú, javaban megírt szoftveren alapul, amit az Internet Archive fejlesztett ki a web archiválására. Ezt a keresőrobotot módosítottam és bővítettem ki. A Heritrixről egy tömör összefoglaló [27]-ben található.

2.6. ábra. **Egy webhely reprezentációja** A webhelyet a tartományneve azonosítja. A relevancia százalék arra egy becslés, hogy a webhelyen található weboldalaknak hány százaléka releváns, azaz tartozik az adott témába. A keresőrobot *várakozási idő* ezredmásodpercenként tölt le a webhelyről egy weblapot. Az webhelyhez tartozó *nyílt csúcsok halmaza* a már felfedezett, de még nem letöltött URL-eket tartalmazza.

2.2.2. A webhely kiválasztási stratégia

A várakozási idő egy pozitív egész. Egy webhelyről *várakozási idő* ezredmásodpercenként töltünk le weblapokat, ahol a várakozási idő dinamikusan változik a webhely relevancia százalékának függvényében.

Hogy meghatározzak egy megfelelő függvényt, bevezetek egy egyszerű modellt. Először is formalizálom a relevancia százalék fogalmát. Minden $H \in$ *webhelyek* webhelyhez tartozik egy $P(o \in R | o \in H)$ valószínűség, annak a valószínűsége, hogy az o weboldal, amit a H webhelyről töltünk le, a releváns weboldalak halmazában, R-ben lesz, ahol R a számunkra érdekes témába



2.7. ábra. **A keresőrobot szerkezete** A keresőrobot működése a webhelyek köré szerveződik. Minden egyes webhelyet önállóan kezelünk. Amikor a webhelyről történt legutolsó letöltés óta eltelt idő eléri a *várakozási időt*, kiválasztunk a webhely *nyílt csúcsok halmazából* a weblap szintű keresési stratégia szerint egy weblapot és átadjuk a *szálkészletnek*. A szálkészletből egy szál hozzárendelődik a weblaphoz, azt letölti és feldolgozza. Attól függően, hogy a weblap releváns volt-e, nő vagy csökken a webhely *relevancia százaléka* és ezzel a *várakozási ideje* is.

tartozó weblapok halmaza. Itt a H maga is egy halmaz, ami a webhelyek halmazába tartozik.

Mint már láttuk, ezt a valószínűséget a következőképpen becsülöm a *relevancia százalékkal*:

$$relevancia\_szazalek_H = \frac{relevans_H}{mind_H} \tag{2.2.2}$$

, ahol a $relevans_H$ a webhelyről eddig letöltött releváns weblapok, a $mind_H$ pedig az összes, a webhelyről letöltött weblap száma.

Egy olyan függvényt keresünk, ami megadja a várakozási időt a relevancia százalék függvényében. Ha egy H_1 webhelyen nincs releváns weblap, azaz ha $relevancia\_szazalek_{H_1} = 0$, akkor a várakozási idő végtelen (legalábbis

nagyon nagy) kell, hogy legyen. Egy H_2, maximális relevancia százalékkal rendelkező webhely esetén ($relevancia\_szazalek_{H_2} = 1$) az ideális érték a 0. Hasonlóan, 0-hoz közeli relevancia százalékokra (pl.: 0.1-re) azt szeretnénk, ha a várakozási idő még mindig nagyon nagy maradna, és 1-től való kis eltérésekre (pl.: 0.9) pedig megmaradna 0 közelében. Ez egy $1/x$-hez, általánosabban pedig egy hatványeloszlás függvényhez hasonló viselkedésű függvényt feltételez.

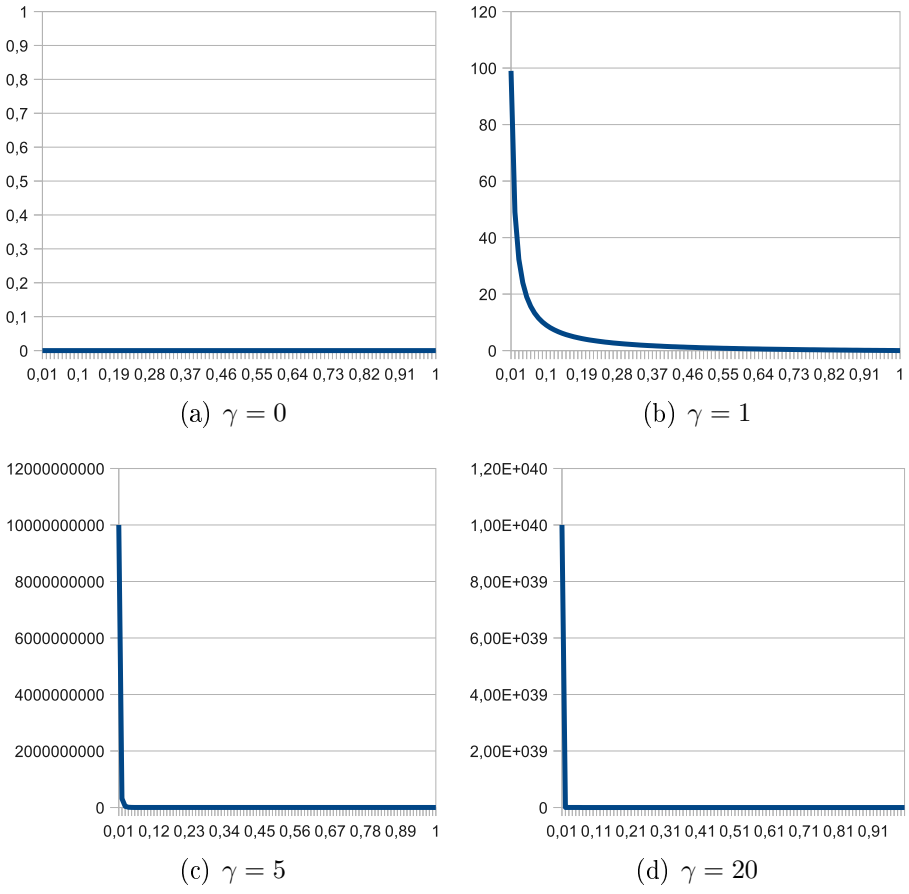
Ha a relevancia százalékot r-rel jelöljük, akkor a várakozási időt kiszámító függvény:

$$f(r) = r^{-\gamma} - 1, \tag{2.2.3}$$

, ahol γ egy paraméter. Az 1-et azért vonjuk ki, hogy $f(1) = 0$ teljesüljön.

A γ paraméterrel azt lehet állítani, hogy mekkora hangsúlyt kapjanak a relevánsabb webhelyek a nem annyira relevánsak ellenében (2.8. és 2.9. ábra). Ha γ nagy, a sok releváns weblapot tartalmazó webhelyeket fogjuk gyakran látogatni a kevésbé relevánsak kárára. Ha γ kicsi, több nem releváns webhelyet is viszonylag gyakran látogatunk, nem hangsúlyozzuk annyira a relevanciát. A szélsőséges eset, ha $\gamma = 0$, ekkor minden webhelyet egyforma sebességgel töltünk le, a relevanciát egyáltalán nem vesszük figyelembe: a keresési stratégia már nem tematikus. Ha γ nagyon nagy (tart végtelenhez), akkor pedig csak a teljesen releváns webhelyeket töltjük: azt, amin akár csak egy nem releváns weblap is van, azt már nem.

Látható, hogy a γ paraméter egyben szabályozza a keresőrobot letöltési sebességét is. Minél nagyobbra állítjuk, annál több releváns webhely kell ahhoz, hogy fenntartsuk a maximális letöltési sebességet. Ha kevés releváns webhely van, és magasra állítjuk, a keresőrobot lelassul. Extrém esetben, ha γ tart végtelenhez, a keresőrobot leáll, mert teljesen releváns webhely nem létezik, legalábbis nagyon ritka a weben (gondoljunk csak a fő– és navigációs



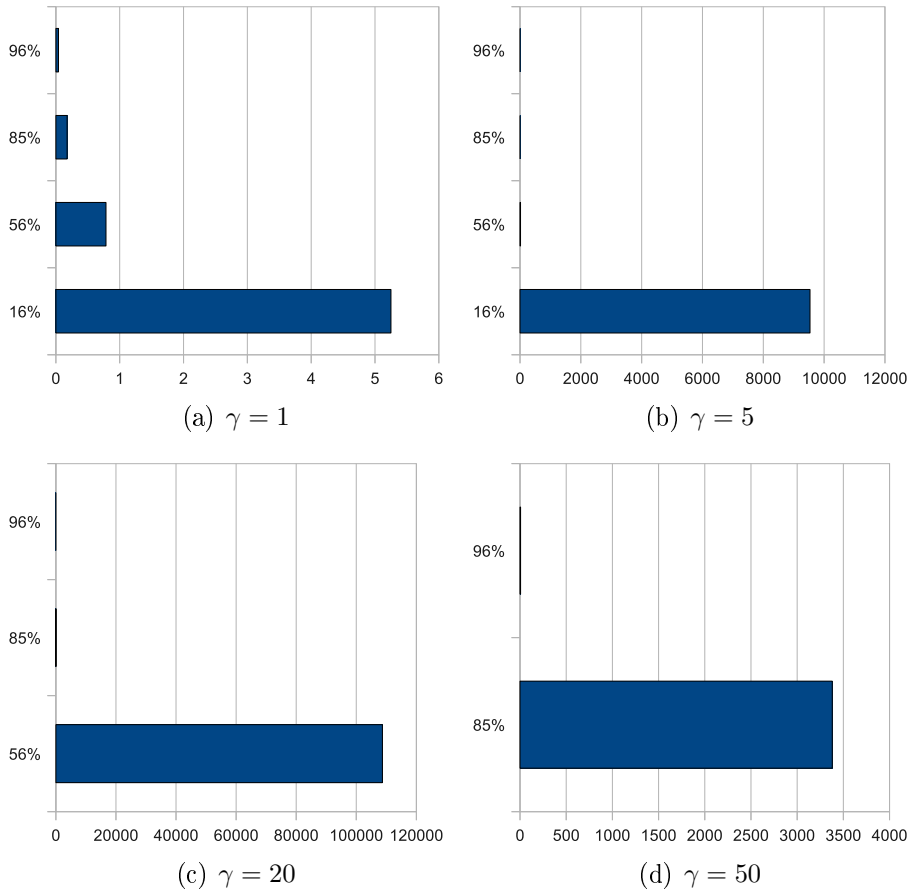
(a) $\gamma = 0$ (b) $\gamma = 1$

(c) $\gamma = 5$ (d) $\gamma = 20$

2.8. ábra. **A várakozási időt kiszámító $f(r) = r^{-\gamma} - 1$ függvény különböző γ paraméterekkel.** Az x tengelyen a relevancia százalék, az y tengelyen az ehhez rendelt várakozási idő látható. $\gamma = 0$-nál egyik webhely sem várakozik, nem tematikus a keresés. Ahogy γ növekszik, egyre nagyobb hangsúlyt kapnak a relevánsabb webhelyek.

oldalakra, amik nem tematikusak). Ha megoldjuk, hogy a keresőrobot ne lassuljon le túlságosan, ez a tulajdonság nagyon előnyös lehet, ugyanis a téma lokalitás elve szerint, ha túl sok nem releváns webhelyet látogatunk meg, akkor az aktív webhelyek listájában elszaporodnak a nem releváns webhelyek. A keresőgép a webnek egy nem releváns területére téved, ahonnan nehéz visszajutni a releváns területekre. Ez elkerülhető, ha a letöltési sebesség ki-

zárólagos optimalizálása helyett a webhelyek relevanciáját is maximalizáljuk.



2.9. ábra. **Különböző relevancia százalékú webhelyekhez tartozó várakozási idők különböző γ paraméterekkel.** Egy 16%-ban, egy 56%-ban, egy 85%-ban és egy 96%-ban releváns webhelyhez tartozó várakozási idő látható különböző γ paramétereknél. Látható, hogy egyre nagyobb relevancia százalék kell ahhoz, hogy a webhelyről töltsünk weblapokat (a nagyon nagy várakozási idejű, nem töltött webhelyeket fokozatosan elhagyom a grafikonokról az áttekinthetőség kedvéért).

Két gyakorlati szempontot is figyelembe kell vennünk f kiszámításakor. Először is ahhoz, hogy $P(o \in R|o \in H)$-t becsülni tudjuk, elegendő mintát kell gyűjtenünk. Ezért amikor egy új webhely kerül az aktív webhelyek listájába, a hozzá tartozó várakozási időt nullára állítjuk mindaddig, amíg nem

gyűjtünk elegendő mintát a becsléshez.

Másodszor, egy T maximális várakozási időt is kell definiálnunk arra az esetre, ha a relevancia százalék nulla lenne. Az $f(r)$ várakozási idő nem lehet nagyobb, mint T. Ha $f(r) > T$, vagy $r = 0$, azaz $r^{-\gamma} = \infty$ lenne, akkor a várakozási időt T-re állítjuk.

2.3. Folyamatvezérlés

A γ folyamatos változtatásával valósítjuk meg a folyamatvezérlést és adaptációt. A barangolás során a gyűjtési rátát és a letöltési sebességet is maximalizálni szeretnénk. A gyűjtési ráta a barangolás egy pontjából visszafele tekintve egy adott időintervallumon a releváns és összes letöltött weboldal számának hányadosa. Tehát a barangolás egy pontján azt adja meg, hogy eddig a pontig bezárólag az elmúlt rövidebb időszakban mennyire jól teljesített a keresőrobot.

Láttuk, hogy a két egymásnak ellentmondó cél közül a gyűjtési ráta maximalizálása fontosabb, ezzel ugyanis meg lehet előzni a keresőrobot eltévedését. Ha a keresőrobot nem téved el, nagy léptékű, több millió weblap letöltését igénylő feladatokat is képes lehet megoldani.

A gyűjtési ráta és a letöltési sebesség egyensúlyát a γ paraméter folyamatos, megfelelő változtatásával érhetjük el. Azt már az előző fejezetben láttuk, hogy ha növeljük γ-t, és ezzel a gyűjtési rátát, akkor a letöltési sebesség csökkenhet. Ha viszont a letöltési sebességet növeljük γ csökkentésével, akkor a gyűjtési ráta eshet le. Ennek a folyamatnak a dinamikája rendkívül függ attól, hogy a keresőrobot éppen a webnek mennyire releváns területén barangol, azaz, hogy mennyire relevánsak a webhelyek az aktív webhelyek listájában. Minél jobban, annál inkább növelhetjük γ-t, és vele együtt a gyűjtési rátát a

letöltési sebesség csökkenése nélkül. Emiatt γ értékét nem rögzíthetjük, hanem folyamatosan változtatni kell az elmúlt időszakban letöltött weblapok relevánssága, azaz a gyűjtési ráta függvényében.

A γ paraméter változtatására a következő módszert alkalmazom: a paraméter értékét a gyűjtési rátához kötöm, viszont, ha nagyon lelassulna a keresőgép, akkor csökkentem γ-t, hogy ne akadjon el.

Ehhez három paraméterre van szükség. Egyrészt meghatározunk a gyűjtési rátának egy *számunkra megfelelő intervallumot*, tehát egy alsó és egy felső korlátot. Ha a ráta az alsó korlát alá megy, akkor növeljük γ-t, ha pedig a felső korlát fölé megy, akkor csökkentjük, így a gyűjtési ráta is nőni ill. csökkenni fog. Ezen kívül definiálunk még egy paramétert, a *maximum várakozási időt két weboldal letöltése között* (ez tehát nem csak egy webhelyre vonatkozik, hanem az egész keresőrobotra). Ha ez alá megy a letöltés sebessége, akkor csökkentjük γ-t, így növeljük a letöltési sebességet. Ez prioritást élvez a gyűjtési ráta szerinti kontroll felett, hiszen ha nem csökkentjük γ-t, nagyon lelassulhat, esetleg le is állhat a keresőrobot.

Meg kell még becsülnünk a *gyűjtési rátát* és a *két weboldal letöltése között átlagosan eltelt időt*, hiszen ezekkel hasonlítjuk össze az előző bekezdésben ismertetett paramétereket. A becsléshez az utolsó n (ahol n egy paraméter) letöltött weblap adatait használjuk fel. A két weboldal letöltése között átlagosan eltelt időt az előző n várakozási idő átlagaként, a gyűjtési rátát pedig az előző n letöltött weboldal közti releváns oldalak relatív gyakoriságaként becsüljük.

2.4. A weblapok osztályozása

Annak eldöntésére, hogy mik a tematikus anyagok, a természetes nyelvfeldolgozás gépi módszereit és felügyelt tanulást használtam. A feladat egy bináris osztályozási feladat: annak eldöntése, hogy egy weboldal az adott témába tartozik-e. A tanítópéldák tehát olyan bemenet-kimenet párok, ahol a bemenet a weboldal, a kimenet pedig egy szám, ami 1, ha a weboldal az adott témába tartozik, és -1, hogyha nem. Természetesen a weboldalak jelentős előfeldolgozáson kell, hogy keresztülmenjenek ahhoz, hogy a felügyelt tanulás fel tudja dolgozni őket; erre szolgálnak a természetes nyelvfeldolgozás gépi módszerei.

A weboldalak feldolgozásával részletesen a 2.4.1 fejezet foglalkozik, itt nagy vonalakban összefoglalom, hogy mi történik a weboldal letöltése után a HTML dokumentummal. Először kivonatoljuk a nyers szöveget, elhagyva a HTML és egyéb kódot, illetve az oldalon található, nem a fő szöveghez tartozó szövegrészeket (pl.: menüpontok, linkek más cikkekre). A szövegben minden szót kicserélünk a szótövére, levágjuk a szavak toldalékait[12]. Így felismerjük például azt, hogy a „macska”, „macskák”, „macskát” szavak mind ugyanazt a fogalmat jelölik. Ezután a szövegből egy szózsák reprezentációt készítünk: a szövegben található szavak gyakoriságán kívül minden információt (pozíció, nyelvtani információk, stb.) elhagyunk. Tehát a szöveget szó–gyakoriság párokkal reprezentáljuk: minden a szövegben szereplő szóhoz hozzátársítjuk előfordulásainak számát. Az utolsó lépésben a szózsákból egy jellegzetességvektort[13] állítunk elő. Minden szóhoz tartozik egy globális, az egész korpuszon érvényes index. A vektor minden eleme megegyezik az elem indexéhez tartozó szó szózsákbeli gyakoriságával. Amelyik indexhez nem tar-

[12] angolul: stemming
[13] angolul: feature vector

tozik gyakoriság a szózsákban (tehát ez a szó nem szerepelt a szövegben), ott 0-t írunk a vektorba. Látható, hogy ezek a vektorok nagy dimenziósak, mert elég sok különféle dokumentum esetén a nyelv legtöbb használt szavához fog index tartozni. Ugyanakkor ritkák is, egy dokumentumban ugyanis a nyelv szavainak csak egy kis része fordul elő, a vektor legtöbb eleme 0 lesz. A jellegzetességvektorok a támogató vektor gép bemenetei.

Mivel felügyelt tanulást használunk, ezért a felhasználónak, aki adott témára szeretné alkalmazni a rendszert, nem kell mást tennie, mint abban a témában weboldalakat gyűjtenie, és gondoskodni arról, hogy a negatív példák között ne legyenek olyanok, amik az adott témába tartoznak. Mivel a negatív példákat tematikus korpuszokból (szöveggyűjteményekből) válogattuk, ez egyszerű feladat: csak a korpuszok a témával foglalkozó részeit kell kivenni a negatív példák közül.

Mint azt később látni fogjuk, a módszer csak egyetlen nyelvfüggő komponenst tartalmaz, azt, ami a szótöveket előállítja a szavakból. Emiatt a módszer más nyelvekre könnyedén alkalmazható, csak ezt a komponenst kell kicserélni.

A felügyelt tanulás algoritmusai közül a választásom a támogató vektor gépekre (a továbbiakban SVM) esett [37]. Joachims mutatta meg először [15], hogy az SVM-ek különösen alkalmasak szövegek kategorizálására. Joachims négy okot sorol fel:

- *Magas dimenziós az input tér*: A szövegekből létrehozott vektor reprezentációk nagy (esetünkben több, mint 100 000) dimenziósak. Az SVM-ek a pozitív és negatív példák közti távolságot maximalizálják, ami védelmet nyújt az overfitting ellen. A védelem hatékonysága nem feltétlen függ a jellemzők[14] számától, ezért az SVM-ek képesek kezelni

[14] angolul: feature

ezeket a magas dimenziós tereket is.

- *Kevés a nem releváns jellemző*: Egy, hagyományos módja az input tér dimenziócsökkentésének annak feltételezése, hogy a jellemzők nagy része irreleváns. Ezen irreleváns jellemzők meghatározása és elhagyása után egy csökkentett dimenziós térben tudunk dolgozni. A szövegkategorizálásban viszont nagyon kevés az irreleváns jellemző: még egy olyan osztályozó, ami csak ezeket az irreleváns jellemzőket használja is sokkal jobban teljesít, mint ha véletlen alapon kategorizálnánk (a részleteket lásd [15]-ben). Tehát a jellemzők elhagyása információ veszteséghez, és az osztályozó hatékonyságának csökkenéséhez vezet.

- *A dokumentum vektorok ritkák*: A dokumentum vektorok kevés nemnulla elemet tartalmaznak.

- *A legtöbb szövegkategorizációs probléma lineárisan szeparálható*: A szövegkategorizálás standard korpuszai az Oshumed [12] és a Reuters-21578 korpusz [9]. Az Ohsumed korpuszban minden kategória lineárisan szeparálható, a Reuters-21578 korpuszban pedig a legtöbb kategória lineárisan szeparálható.

Joachims eredményei megmutatták, hogy az SVM-ek a szövegkategorizációs algoritmusok élvonalába tartoznak. Számunkra nem csak az algoritmus jósága, hanem sebessége is számít. A keresőrobotban az osztályozó sebessége szűk keresztmetszet, hiszen rengeteg weboldalt töltünk le és dolgozunk fel. Ezért, és mivel láttuk, hogy a legtöbb szövegkategorizációs probléma lineárisan szeparálható, az SVM-nek egy gyors, lineáris változatát használjuk.

Összefoglalva, a [14]-ben leírt lineáris támogató vektor gépet használtam duális koordináta ereszkedés módszerrel és L_2 veszteségi függvénnyel.

Az SVM bináris osztályozó, ami eldönti, hogy egy dokumentum a témába tartozik-e.

A következőkben részletesen leírom a jellegzetességvektorok előállításának és a támogató vektor gép tanításának folyamatát.

2.4.1. A jellegzetességvektorok előállítása

A letöltött weblapokat több lépésben alakítjuk át jellegzetességvektorokká (2.10. ábra).

Az első lépés a HTML dokumentumból a szöveg kinyerése. Ez azt jelenti, hogy megtisztítjuk a HTML dokumentumot a különböző, nem megjelenített tartalmaktól (pl.: HTML-elemek, JavaScript), és a megjelenített, de nem az oldal fő szövegéhez tartozó szövegelemektől (pl.: menük, dátum). A HTML fájlból elemzéssel előállítom a *Document Object Model*-t, ami a HTML dokumentum egy hierarchikus reprezentációja. Az elemzéshez egy nyílt forrású eszközt, a HtmlCleanert[15] használom.

Ezután egy saját fejlesztésű, egyszerű algoritmussal kinyerem a szöveget a HTML dokumentumból. Az algoritmus bejárja a DOM fa minden csúcsát. Azokat a csúcsokat, amik nem szöveget, vagy JavaScriptet tartalmaznak kihagyja. A szöveg típusú csúcsokat megszűri. Az egy előre meghatározott konstansnál (100-nál) kevesebb karaktert tartalmazó szövegrészeket és az alfanumerikus karaktereket 80%-nál kisebb arányban tartalmazóakat eldobja. Az előbbi a menüpontokat dobja el, illetve a linkeket más oldalakra. Az utóbbi a JavaScriptek szűrése után fennakadt kódokat dobja el. A konstansokat tapasztalati úton állítottam be. A HTML oldalból kinyert szöveg az algoritmus által nem eldobott szövegrészek konkatenációja. Előfordulhat, hogy a szövegrészek sorrendje megváltozik az eredeti HTML oldalon látható

[15] `http://htmlcleaner.sourceforge.net/`

sorrendhez képest, ez azonban nem okoz problémát, mivel ezt az információt később eldobjuk.

A második lépés a szavak kicserélése szótövekre a kinyert szövegben. Erre a célra a Porter Stemmert [34] használom, ami egy alapvető stemmer algoritmus a természetes nyelvfeldolgozásban. Habár egy szótő különböző toldalékokkal fordulhat elő, ezeket a toldalékolt formákat a szótőképző[16] ugyanarra a szótőre képezi le. Így a következő lépésekben képesek vagyunk a toldalékolt formákat egységesen kezelni: például a „macskának", „macskák", „macskát" szavak helyett mindenhol a „macska" szótővel dolgozunk. Ez egyrészt jelentősen csökkenti az input tér dimenzióját, másrészt javítja az osztályozó teljesítményét, ugyanis szótő keresés nélkül az osztályozó nem képes felismerni, hogy a három toldalékolt alak ugyanarra a fogalomra vonatkozik.

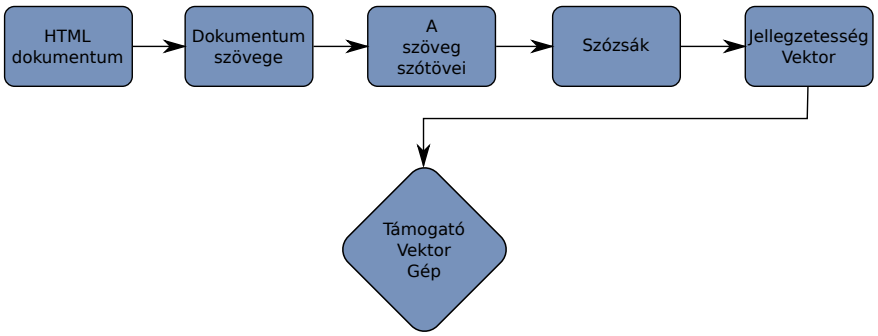
A harmadik lépésben elkészítjük a szózsák reprezentációkat. A szavak sorrendje, a mondatszerkezetek, bekezdések, egyszóval az egész szöveg struktúrája elvész, csak az egyes szavak gyakoriságát tartjuk meg. Megszámoljuk minden egyes szótő előfordulásainak számát, és egy szótő–gyakoriság párokból álló reprezentációt készítünk. Egy ilyen (sz_i, n_i) párban w_i egy egyedi szótő és n_i ennek a szótőnek a gyakorisága a szövegben. A legtöbb szövegfeldolgozó algoritmus ezt a reprezentációt használja, mert elég információ marad meg belőle ahhoz, hogy utalni tudjon a szöveg értelmére, az elveszett struktúrát pedig nehéz és időigényes lenne hasznosítani. Esetünkben a rendszer sebessége kritikus szempont, ezért választottam ezt a reprezentációt.

A negyedik, és egyben utolsó lépés a jellegzetességvektorok elkészítése. Ehhez szükséges egy korpusz, én a tanítópéldákat használom. Lényegében összesítjük a tanítópéldák szózsákjait. Erre azért van szükség, mert az SVM vektorokat vár bemenetként, nem szótő–gyakoriság párokat. Az összesítés

[16]angolul: stemmer

első lépéseként végigmegyünk a mintákon, és minden szótőhöz rendelünk egy egyedi indexet, 0-tól indulva, egyesével növelve. Tehát ha olyan szótövet találunk, amit még nem láttunk, akkor hozzárendeljük a legutóbbi indexet, majd növeljük egyel.

A legnagyobb index plusz egy lesz az input tér és egyben a jellegzetességvektorok dimenziója. Egy dokumentumhoz a következőképpen rendelünk egy jellegzetességvektort: végigmegyünk a hozzá tartozó szózsákon, és minden szótőhöz megkeressük a hozzá tartozó indexet. Majd a jellegzetességvektorban beállítjuk az adott indexű elemet a szótő gyakoriságára. A többi index nulla marad. Előfordulhat, hogy egy frissen letöltött HTML dokumentum esetében lesz olyan szótő, amihez nem tartozik index. Ekkor ezt a szótövet eldobjuk, nem vesszük figyelembe. Ezzel elegendő sok tanítópélda esetén kevés információt veszítünk, sőt, szűrésnek is tekinthető. A jó kísérleti eredmények is ezt bizonyítják.



2.10. ábra. **A letöltött weblapok osztályozása** A letöltött weblapból (HTML dokumentumból) első lépésben kivonatoljuk a szöveget, majd a szavakat szótövükre cseréljük. Ebből a struktúra eldobásával és a gyakoriság megtartásával egy szózsák reprezentációt készítünk. A szózsákot egy jellegzetességvektorra képezzük le. Minden szótőhöz tartozik egy jellegzetességvektorbeli index, a vektorban ezt az indexű elemet beállítjuk a szótő szózsákbani gyakoriságára. A támogató vektor gép ezt a jellegzetességvektort kapja meg, ami alapján eldönti, hogy a weblap az adott témába tartozik-e.

2.4.2. A támogató vektor gép tanítása

Az SVM tanításához a mintákat öt korpuszból gyűjtöttem. A pozitív (témába tartozó) tanítópéldák filmkritikák az IMDB (Internet Movie Database) rec.arts.movies.reviews hírcsoport archívumából[17]. Tehát a választott téma, amiről szóló dokumentumokat gyűjtünk, a „filmek".

A negatív példák gyűjtése nehezebb volt, hiszen ezeknek le kell fednie az összes nem filmekkel kapcsolatos témát, amivel a weben találkozhatunk. Választásom különböző témájú, szövegfeldolgozásban használatos korpuszokra esett. A négy felhasznált korpusz, és elérhetőségük a dolgozat írásának idején:

1. A Reuters-21578 korpuszt széles körben használják szövegkategorizációs kísérletekhez. Főleg gazdasági, pénzügyi témájú cikkeket tartalmaz a Reuters archívumából.

   `http://www.daviddlewis.com/resources/testcollections/reuters21578/`

2. A „4 Universities Data Set" egyetemek informatika tanszékeinek weboldalait tartalmazza.

   `http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/`

3. A „20 Newsgroups Data Set" kb. 20 000, hírcsoportokból származó dokumentumot tartalmaz, megközelítőleg egyenlő arányban szétosztva 20 különböző téma között.

   `http://people.csail.mit.edu/jrennie/20Newsgroups/`

4. Az „Industry Sector Data Set" 70 különböző témában tartalmaz dokumentumokat.

[17] `http://www.imdb.com/Reviews/`

`http://www.cs.umass.edu/~mccallum/data/sector.tar.gz`

Összesen 28 000 pozitív és 56 000 negatív példát gyűjtöttem. Mielőtt felhasználásra került volna, önmagában is teszteltem az osztályozót. Egy 10-szeres keresztvalidációt végeztem a $28000 + 56000 = 84000$ tanítópéldán, és kiszámítottam a *pontosság* és *lefedettség* értékeket, ahol a

$$pontossag = \frac{helyesen\ pozitivnak\ osztalyozott\ dokumentumok\ szama}{minden\ pozitivnak\ osztalyozott\ dokumentum} \tag{2.4.1}$$

$$lefedettseg = \frac{helyesen\ pozitivnak\ osztalyozott\ dokumentumok\ szama}{osszes\ valoban\ pozitiv\ dokumentum} \tag{2.4.2}$$

A *pozitív* itt témába tartozót jelent.

Eredményeim meglepően jók lettek. A pontosság 0.993, a lefedettség 0.998 lett. Az ebből számolt *F-mérték*, ami a kettő harmonikus átlaga, 0.995.

Mivel ezek az értékek az irodalomban ismertetettekhez képest (lásd [15]) is meglepően jónak tekinthetők, egy további tesztet is elvégeztem. Véletlenszerűen választottam és letöltöttem a webről 15 filmkritikát és 15 egyéb weboldalt (általános híreket), és osztályoztam őket a már betanított SVM-mel. Azt tapasztaltam, hogy az SVM kivétel nélkül mindegyik dokumentumot helyesen osztályozta.

A jelenségnek két lehetséges okára szeretnék rávilágítani. Egyrészt nagyon sok tanítópéldát gyűjtöttem és használtam, egy–kettő nagyságrenddel többet, mint ami az irodalomban szokásos. Ez pozitívan befolyásolja az SVM működését, az overfitting pedig a margó maximalizálás miatt nem következik be. Másrészt valószínű, hogy a feladat, mint tanulási feladat könnyen tanulható az SVM számára: a két mintahalmazt lineárisan szeparálható, és az elválasztó hipersík nagy margóval képes elválasztani őket.

2.5. Eredmények

A kísérletek fő célja annak a kérdésnek a megválaszolása, hogy képes-e a keresőrobot a megadott témába tartozó weblapokat gyűjteni és webhelyeket felderíteni. Ha jól teljesít, az egyben a webhely lokalitás elv használhatóságának is egy bizonyítéka.

Mivel a módszer következő fázisához, a legbefolyásosabb webhelyek meghatározásához nagyon nagy számú letöltött weblapra van szükség, az is fontos kérdés, hogy a keresőgép képes-e nagy léptékű barangolásra. A nagy léptékű tematikus barangolás nehéz feladat, ugyanis minél tovább tart a keresés, annál nagyobb területen kell dolgoznia a keresőrobotnak, így nagyobb az esély arra, hogy nem releváns területre téved, ahonnan nehéz visszatalálni. Különösen a felügyelt tanulást a keresési stratégia részeként használó keresőrobot esetén áll fenn a veszélye annak, hogy olyan területre jut, amit a tanítópéldák nem írnak le megfelelőképpen, ahol aztán „eltéved".

Keresőrobotom több szempontból is védett az eltévedés veszélyétől. A webhely szerinti dekompozíciós elv miatt nem weboldalról weboldalra ugrik, hanem egy globális adatbázisa van az elérhető webhelyekről, és ebből választja ki minden lépésben a következő webhelyet, amiről egy weboldalt letölt. A folyamatvezérlés megakadályozza, hogy ez az adatbázis (az aktív webhelyek listája) nem releváns webhelyekkel töltődjön fel. A keresési stratégiában pedig nem használok felügyelt tanulást: az osztályozóm csak egy dokumentumot kap bemenetnek és azt dönti el, hogy ez a témába tartozik-e vagy sem, az eredményeket pedig ezután összesítem webhely szerint.

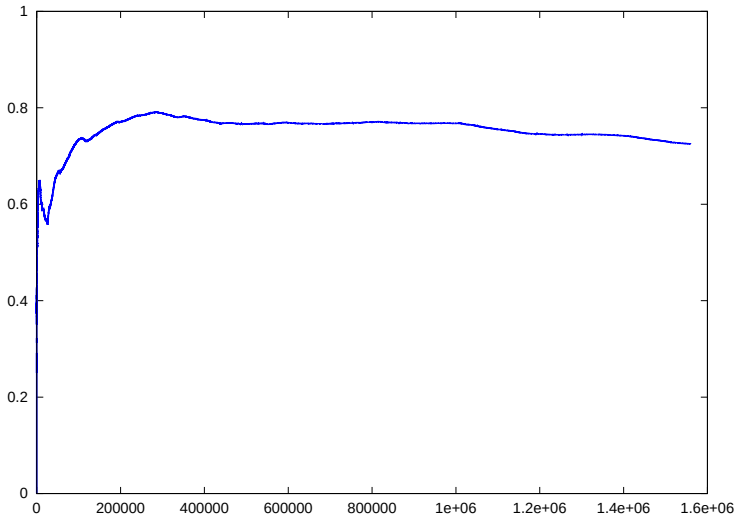
A keresőrobot teljesítményét a *gyűjtési rátával*, a tematikus keresőrobotok jóságának elfogadott mércéjével [3] mérjük. Ez n weblap letöltésére visszatekintve a releváns összegyűjtött weblapok relatív gyakorisága. n-et 100-nak választottam.

webhely	relevancia százalék
stickyfloor.blogdrive.com	1.0
moviesblog.mtv.com	1.0
moviepalace.blogspot.com	0.99
adnauseum.blogdrive.com	0.99
www.combustiblecelluloid.com	0.98
vergingwriter.blogspot.com	0.98
distantorigin.blogdrive.com	0.97
jadedviewer.com	0.97
www.greencine.com	0.95
www.reelviews.net	0.95
filmfreakcentral.blogspot.com	0.95
amusicment.blogspot.com	0.95
criticalcorner.net	0.95
www.dvddrive-in.com	0.94
www.thefilmpanelnotetaker.com	0.94
www.breabennett.name	0.94
ferdyonfilms.com	0.92
www.arabfilm.com	0.92
www.horrorwatch.com	0.92
mrpeelsardineliqueur.blogspot.com	0.91

2.1. táblázat. **A húsz legrelevánsabb webhely.**
A legtöbb webhelyről első pillantásra látszik, hogy filmekkel foglalkozik. Az alaposabb ellenőrzés során kiderült, hogy mindegyik a témába tartozik.

Egy másik mércét is használok, a releváns weblapok relatív gyakoriságát az eddig összesen gyűjtött weblapok közt, ezt *kumulatív gyűjtési rátának* nevezem. Ennek értéke a barangolás végén megadja az összesen gyűjtött releváns weblapok relatív gyakoriságát.

Néhány próbafuttatás után a keresőrobot paramétereit a következő értékekre állítottam. A *hivatkozásszám küszöb* egyenlő 200–zal, tehát egy webhelyet akkor teszünk be az aktív webhelyek listájába és kezdjük letölteni, ha már legalább 200-szor hivatkoztak rá a listából. Először kisebb értékekkel kísérleteztem, de az aktív webhelyek listája drámaian növekedett. Emi-

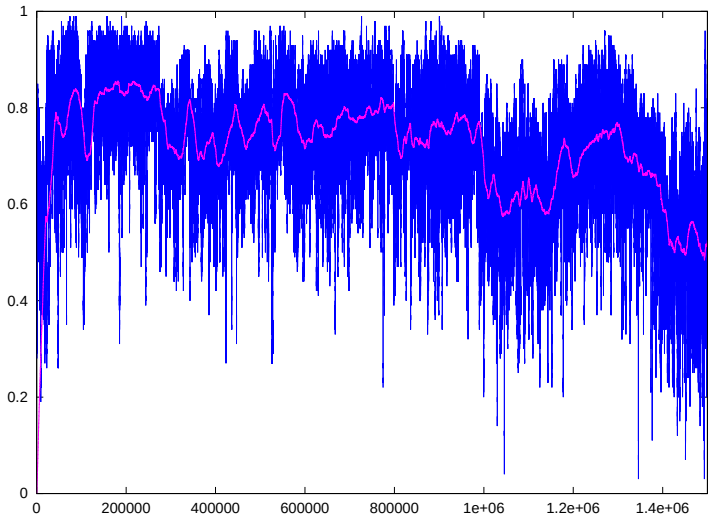


2.11. ábra. **A kumulatív gyűjtési ráta a letöltött weblapok számának függvényében, I. futtatás.** Látható, hogy a *kumulatív gyűjtési ráta* (y tengely) folyamatosan magas és majdnem konstans marad ahogy a letöltött weboldalak száma (x tengely) növekszik, egészen 1.6 millió letöltött weblapig. A grafikon készítésénél simítást nem alkalmaztam, mind az 1.6 millió adatpont szerepel.

att, mivel minden egyes hozzáadott webhelyről le kell tölteni egy minimális mennyiségű weblapot ahhoz, hogy becsülni tudjuk a webhely relevancia százalékát, a gyűjtési ráta is leesett. A 200-as érték elég nagy ahhoz, hogy nem releváns webhelyek ne nagyon kerüljenek be az aktív webhelyek listájába, és elég kicsi ahhoz, hogy ne zárjuk ki a valóban releváns webhelyeket a keresésből.

Az egy webhelyről a relevancia százalék becsléséhez gyűjtendő minták számát 10-re állítottam. Ha nagyobb számban vannak releváns weboldalak egy webhelyen, akkor a kezdeti relevancia százalék elég nagy lesz 10 minta gyűjtése után is ahhoz, hogy elindulhasson a webhely letöltése. A letöltés folyamán pedig folyamatosan javul a becslés.

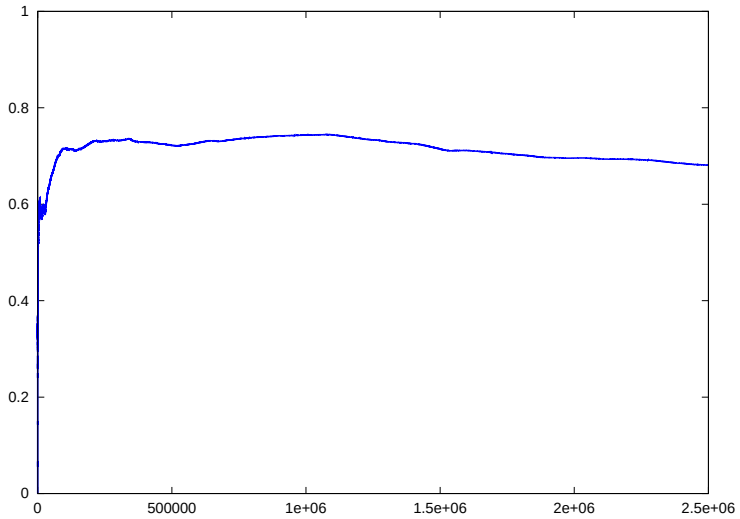
Mivel a keresőrobot maximálisan elérhető teljesítményére voltam kíván-



2.12. ábra. **A gyűjtési ráta a letöltött weblapok számának függvényében, I. futtatás.** A *gyűjtési ráta* folyamatosan magas marad ahogy a letöltött weboldalak száma (x tengely) növekszik, egészen 1.6 millió letöltött weblapig. A gyűjtési ráta egy mozgó átlagát lilával jelöltem. A grafikon készítésénél simítást nem alkalmaztam, mind a 1.6 millió adatpont szerepel.

csi, ezért az *elérendő gyűjtési ráta alsó és felső korlátját* is magasra állítottam, $0,8$-ra illetve $0,9$-re. A *maximális várakozási időt két weblap letöltése között* úgy állítottam be, hogy a keresőrobot legalább három weboldalt töltsön le másodpercenként. A *gyűjtési ráta* és a *két weboldal letöltése között átlagosan eltelt idő* becsléséhez n-et 100-ra állítottam. A γ paramétert konstans amplitúdóval, $\pm 0,01$–gyel frissítem minden weboldal letöltése után. T, a maximális várakozási idő 8 óra, tehát egy webhelyről legalább 8 óránként letöltünk egy weblapot.

A „filmek”, ezen belül a „filmkritikák” témában gyűjtünk weblapokat, tehát pontosan azok a webhelyek és weblapok relevánsak, amik filmekről szólnak. A téma mellett szólt népszerűsége és az, hogy a weben nagyon sok weboldal van és keletkezik ebben a témában.

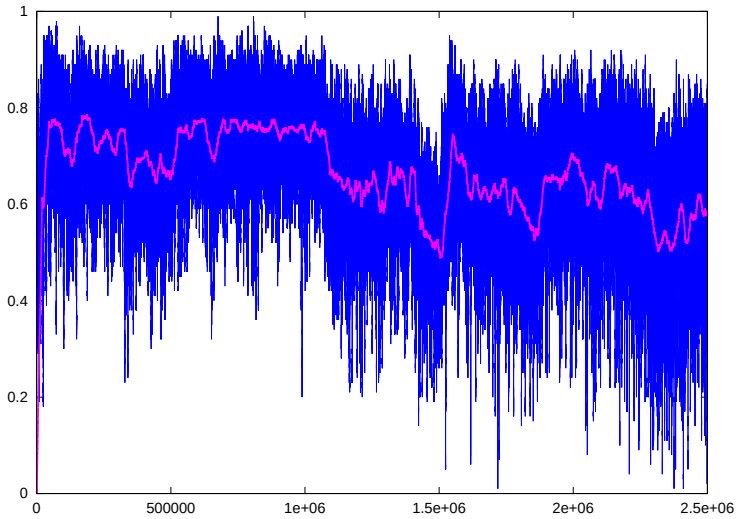


2.13. ábra. **A kumulatív gyűjtési ráta a letöltött weblapok számának függvényében, II. futtatás.** Látható, hogy a *kumulatív gyűjtési ráta* (y tengely) folyamatosan magas és majdnem konstans marad ahogy a letöltött weboldalak száma (x tengely) növekszik, egészen 2.5 millió letöltött weblapig. A grafikon készítésénél simítást nem alkalmaztam, mind a 2.5 millió adatpont szerepel.

A keresőrobot inicializálásához szükséges kezdeti webhely lista webcímei a „Best of the Web" bloggyűjteményből származnak, az „Arts/Movies/Reviews" kategóriából[18]. A kezdeti webhely listába így csak 17 webcím került. A futtatások folyamán a keresőrobot több, mint 4000 a témába tartozó webhelyet talált csupán ebből a 17-ből kiindulva.

Kísérleteimben a keresőrobot az irodalomban elért eredményekhez képest[19] is kimagasló teljesítményt mutatott. A két futtatásban összesen több, mint 4 millió weblapot töltött le, a kumulatív gyűjtési ráta pedig folyamatosan 0.7 fölött maradt (2.11. és 2.13. ábra), így a futtatás végére a letöltött weblapok több, mint 70%-a volt releváns. Átlagosan négy weblapot töltött

[18] `http://blogs.botw.org/Arts/Movies/Reviews/`
[19] lásd: 2.1. fejezet irodalmi hivatkozásai



2.14. ábra. **A gyűjtési ráta a letöltött weblapok számának függvényében, II. futtatás.** A *gyűjtési ráta* folyamatosan magas marad ahogy a letöltött weboldalak száma (x tengely) növekszik, egészen 2.5 millió letöltött weblapig. A gyűjtési ráta egy mozgó átlagát lilával jelöltem. A grafikon készítésénél simítást nem alkalmaztam, mind a 2.5 millió adatpont szerepel.

le másodpercenként. A gyűjtési ráta (2.12. és 2.14. ábra) ingadozik, viszont hosszabb távon (lila vonal) a fluktuációk eltűnnek, a ráta pedig magas marad.

A 2.1. táblázatban a 20 legrelevánsabb felfedezett webhely látható. A legtöbbről első pillantásra látszik, hogy filmekkel foglalkozik. Az alaposabb ellenőrzés során kiderült, hogy mindegyik a keresett témába tartozik.

2.6. Összefoglalás

Ebben a fejezetben a téma lokalitás fogalmát kiterjesztettem. A webhely lokalitás elve szerint az egy webhelyen található weboldalak hasonló témáról szólnak. Erre építve terveztem és megvalósítottam egy tematikus adaptív keresőrobotot.

A keresőgéppel végzett kísérletek azt bizonyítják, hogy képes nagy léptékű, több millió dokumentum letöltésével járó keresések hatékony lebonyolítására. A filmek témakörében 4,1 millió dokumentumot töltöttem le, ezeknek több, mint 70%-a tartozott a témába.

A keresőrobot képes a releváns webhelyek felderítésére és anyagok gyűjtésére egy kijelölt témában. A téma kijelöléséhez nincs szakértő tevékenységre szükség, csupán az adott témába tartozó dokumentumok szükségesek. Emiatt a portálkészítést hatékonyan segíthetjük bármilyen témában.

3. fejezet

A legnagyobb befolyással bíró webhelyek kiválasztása

3.1. Bevezetés

A tematikus portálok fő célja a legfontosabb hírek és webhelyek összegyűjtése egy adott témában. Az előző fejezetben láttuk, hogy a webhely alapú keresőrobot képes megkeresni a legrelevánsabb webhelyeket, tehát azokat, amik főleg az adott témával foglalkoznak. De ahhoz, hogy egy webhely egy portálra kerülhessen, nem csak az kell, hogy a megfelelő témával foglalkozzon, hanem, hogy az adott témában meghatározó szerepe legyen. Hiába foglalkozik egy webhely kizárólag filmekkel, ha senki sem olvassa, nem tartalmaz érdekes híreket, nincs befolyása, akkor nem kerül fel portálokra. Az, hogy egy webhely befolyásából a hasznosságára lehet következtetni, már régóta elfogadott az webes keresők világában, gondoljunk csak a Google PageRank algoritmusára [29].

A webhelyek befolyásának meghatározására a PageRank -nél jobb megközelítések is léteznek. Ebben a fejezetben egy ilyen megközelítést mutatok

be, és alkalmazom a keresőrobotom által letöltött webhelyekre.

Az előző fejezetben bemutatott keresőgép a tematikus barangolás során egy gráfot készít a világháló általa feltérképezett részéről. Minden egyes weblap feldolgozása során a weblap webcíme, az összes a weblapról hivatkozott weboldal webcíme, és az, hogy a weblap releváns-e, egy fájlba íródik. A barangolás befejeztével összeállítjuk ebből a gráfot, ahol a csúcsok a releváns weboldalak webcímükkel reprezentálva, az élek pedig az őket összekötő hiperhivatkozások. Ezután erre a gráfra alkalmazunk egy, a szubmodularitáson alapuló eljárást, amit Leskovec [20] alkalmazott először egy blog korpuszra. Én az eljárást egy automatikusan, keresőrobottal gyűjtött adathalmazra, webhelyekre alkalmazom.

Az eljárás alapgondolata az, hogy a hírek terjednek a világhálón, így egy hírhez rendelhető egy csúcshalmaz a gráfból, ami a hírt közlő weblapokat tartalmazza. Egy ilyen csúcshalmazt információs kaszkádnak [2] nevezünk. A célunk egy portálon az, hogy minél több hírt, információs kaszkádot detektáljunk. Azt, hogy egy webhely mennyi befolyással bír, az általa detektált kaszkádok számával jellemezhetjük. Mivel a felhasználó által kezelhető, tehát a portálra kitehető webhelyek száma véges, csak n darab webhelyet tudunk kiválasztani.

A kérdés tehát az, hogy melyik n webhelyet kell kiválasztanunk ahhoz, hogy a lehető legtöbb kaszkádot lefedjék. Mint látni fogjuk, erre az egyébként nehéz kombinatorikus optimalizálási feladatra a feladat szubmodularitásának köszönhetően egy mohó algoritmus az optimális eredmény legalább 63%-át képes elérni. Más szóval matematikai garanciák (tételek) vannak arra, hogy eredményünk nem rosszabb, mint az elérhető legjobb eredmény 63%-a.

3.2. A szubmodularitás fogalma

Az elmúlt években rohamos fejlődés ment végbe a kombinatorikus optimalizálás, ezen belül a szubmoduláris függvények maximalizálásának alkalmazásai terén. Ez a fejlődés főleg a módszer gyorsaságának köszönhető, és annak, hogy elméleti garanciát nyújt a legrosszabb lehetséges megoldás jóságára nézve.

A módszer 30 éves múltra tekint vissza. Matematikusok fejlesztették ki [11], a gépi tanulással foglalkozók körében pedig 2003-ban vált ismertté Kempe és mások [17] munkája nyomán, ami a befolyás maximalizálásáról szólt társadalmi hálózatokban.

Anélkül, hogy az elmélet részleteibe belemennék, megadom a szubmoduláris függvény definícióját és intuitív jelentését. Egy halmazokon értelmezett $F : 2^V \to \mathbb{R}$ függvény *szubmoduláris*, ha bármely $A \subseteq B \subseteq V$-re és bármely $s \in V \setminus B$-re

$$F(A \cup \{s\}) - F(A) \geq F(B \cup \{s\}) - F(B) \tag{3.2.1}$$

Ennek egy intuitív jelentése, ha a V halmaz elemeit valamilyen szenzoroknak tekintjük, az F függvény pedig a V halmaz részhalmazaihoz szenzoros értékeket rendel. Ekkor az F szubmodularitása annak felel meg, hogy ha kevesebb szenzorhoz (A halmaz) veszünk hozzá egy új szenzort, azzal több információt nyerünk, mint ha az ezeken a szenzorokon felül még plusz szenzorokat tartalmazó B halmazhoz vesszük ugyanezt a szenzort.

3.3. Az algoritmus bemutatása

Első lépésként formalizáljuk a feladatot. A keresőrobot futásának eredménye egy $G = (V, E)$ irányított gráf, ahol V, a csúcsok halmaza, a releváns weblapok webcímeit, E, az élek halmaza pedig az őket összekötő hiperhivatkozásokat tartalmazza. Egy hír akkor terjed egy weblapról egy másikra, ha az utóbbi linkel az előzőre. Tehát az információ terjedés iránya pontosan ellentétes a hiperhivatkozások irányával. Ezért egy új, $G' = (V, E')$ információ terjedés gráfot készítünk a G-beli élek megfordításával.

Egy információs kaszkád Leskovec–féle definíciója a következő. A kaszkád a hozzá tartozó hír eredeti forrásából, egy G'-beli csúcsból indul ki. A hír G' élein terjed, egy információs kaszkádot tehát úgy találunk meg, hogy elindulunk az eredeti forrásból, végigmegyünk G' élein, és az e közben talált csúcsokat belevesszük a kaszkádba.

Minden webhelyhez tartozik egy csúcshalmaz, ami a webhelyen található weblapokat tartalmazza. A feladatunk n darab webhelyet kiválasztani az összes webhely közül úgy, hogy a halmazban található weblapok a lehető legtöbb kaszkádot lefedjék. Egy weblap lefed egy kaszkádot, ha a kaszkád egyik csúcsa a weblap. Más szóval ha legalább egy olyan webhelyet kiválasztottunk, aminek az egyik weboldala benne van az információs kaszkádban, tehát a kaszkádhoz tartozó hírt közli, a kaszkádot lefedettnek tekintjük.

A formalizált feladat a következő. A G' gráfon kiválaszthatunk egy A csúcshalmazt, amik lefednek valahány kaszkádot. Csak egész webhelyeket választhatunk ki, tehát egy lépésben az A halmazt csak egy webhely összes weblapjával növelhetjük. Az $R(A)$ jutalomfüggvény az A elhelyezés jóságát adja meg, ami a lefedett kaszkádok száma. Ezt a függvényt szeretnénk maximalizálni a G' gráf V csúcshalmazán, azzal a feltétellel, hogy csak n webhelyet választhatunk.

Minden webhely ugyanannyi helyet foglal el egy portálon, így egy h webhely költsége $c(h) = 1$. $R(A)$-t szeretnénk maximalizálni a $c(A) \leq n$ feltétel mellett, ahol $c(A)$ az A halmazban lévő webhelyek száma. $R(A)$-ról meg lehet mutatni, hogy szubmoduláris, azaz minden $A \subseteq B \subseteq V$ elhelyezésre és $h \in V \setminus B$ webhelyre igaz, hogy

$$R(A \cup \{h\}) - R(A) \geq R(B \cup \{h\}) - R(B). \tag{3.3.1}$$

Habár a szubmoduláris függvények maximalizálása NP-nehéz [18], meg lehet mutatni, hogy esetünkben, mivel minden webhely költsége 1, a következő mohó algoritmus közel-optimális [11], azaz az optimum legalább 63%-át legalább eléri.

Az üres A_0 elhelyezéssel kezdünk, és iteratívan haladunk. A k. lépésben azt a h_k webhelyet adjuk hozzá A_{k-1}-hez, ami a maximális növekedést biztosítja.

$$h_k = \arg \max_{h \in V \setminus A_{k-1}} R(A_{k-1} \cup \{h\}) - R(A_{k-1}) \tag{3.3.2}$$

Az algoritmus akkor áll meg, ha n webhelyet kiválasztott. Látható, hogy n–től az algoritmus menete nem függ, n csupán azt dönti el, hogy meddig tart az iteráció. Így nem szükséges egy rögzített n-t választani előre, az algoritmust egyéb feltételekkel is le lehet állítani, mint például a detektált kaszkádok aránya az összeshez viszonyítva.

3.4. Eredmények

Kísérleteimet a keresőrobotom egy futása alatt tematikusan gyűjtött több, mint 4000 webhelyre és ezeken kb. 1.6 millió weblapra alapoztam. A G' gráfot az előző fejezetben vázolt módon állítottam elő.

A G' gráfban a nem triviális (kettő csúcsnál többet tartalmazó) információs kaszkádokkal dolgoztam. Az információs kaszkád már említett Leskovec–féle definícióján [20] módosítottam annyiban, hogy az elosztókat[1] kizártam a kaszkád meghatározásakor, más szóval azokon a G'–beli éleken, amik gyűjtőlapokba vezetnek vagy onnan kivezetnek, nem mentem végig. Az elosztók (esetemben a több, mint 200 kimenő hiperhivatkozással rendelkező weblapok) ugyanis nem egy hírt vagy cikket tartalmaznak, viszont sok hírt tartalmazó weblapra linkelnek, így elrontják a kaszkádokat. A mindennapi tapasztalaton felül ez kísérleteim alapján is látható volt: az elosztók használata esetén több valószínűtlenül nagy kaszkád keletkezett, 10^6, 10^7 darab csúccsal. Az elosztók kizárása után ezek eltűntek.

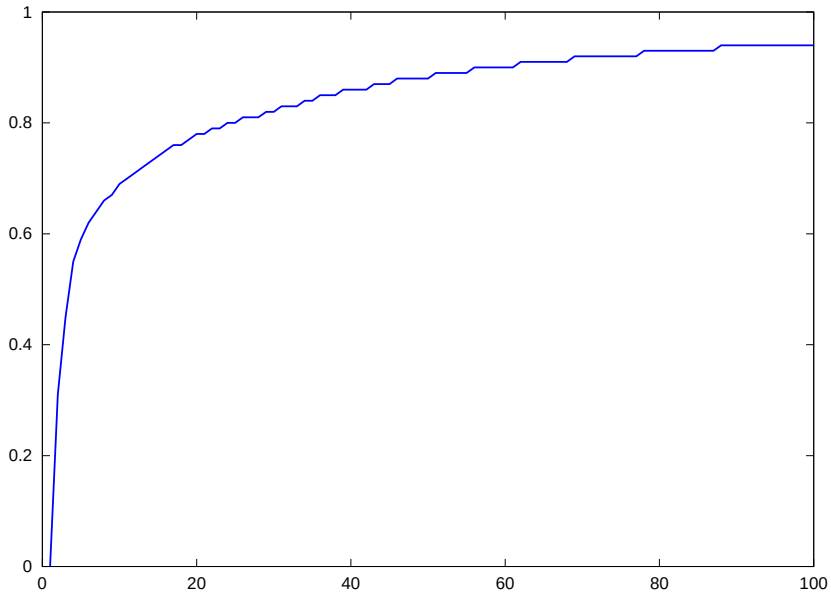
A költségvetést $n = 100$-ra állítottam. Mint említettem, az algoritmus minden közbenső lépésben is közel-optimális eredményt ad. A 3.1. ábrán látható a megtalált kaszkádok aránya az összes kaszkádból minden lépésben. Eredményeim Leskovec eredményéhez hasonlóan azt mutatják, hogy már nagyon kevés (20–40) webhely lefedi a kaszkádok nagyon nagy részét (80–90 százalékát). Ez azt jelenti, hogy képesek vagyunk felsorolni egy témában egy kicsi, 20–40 webhelyből álló listát, ami tartalmazza majdnem az összes tartalmat ebben a témában.

A 3.1. táblázatban felsorolom az első 20, az algoritmus által talált webhelyet, az általuk detektált további kaszkádok számát, és a keresőgép által hozzájuk rendelt relevancia százalékot. A webhely által detektált további kaszkádok száma a webhely hozzávétele után és az algoritmus előző lépésében detektált kaszkádok különbsége. Látható, hogy a legbefolyásosabb webhelyek nem egyeznek meg a legrelevánsabbakkal. Az 2.1. táblázatban szereplő 20 legrelevánsabb webhely csupán 523 kaszkádot detektál.

[1] angolul: hub

webhely	lefedett kaszkádok	relevancia
daily.greencine.com	3093	0.77
www.impawards.com	1380	0.89
slate.msn.com	991	0.46
oggsmoggs.blogspot.com	415	0.80
opalfilmsarchive.blogspot.com	314	0.56
carson83.blogspot.com	162	0.69
bleeding-tree.blogspot.com	159	0.41
templeofschlock.blogspot.com	158	0.46
www.geocities.com	148	0.42
metaphilm.com	125	0.52
kamikazecamel.blogspot.com	114	0.48
diyfilmmaker.blogspot.com	102	0.40
actionflickchick.com	93	0.84
mysterymanonfilm.blogspot.com	87	0.83
blogs.amctv.com	84	0.57
ferdyonfilms.com	76	0.92
www.chutry.wordherders.net	70	0.65
he-shot-cyrus.blogspot.com	66	0.48
www.premiumhollywood.com	61	0.56
www.horror-101.com	61	0.67
Összes detektált kaszkád	7759	

3.1. táblázat. **A 20 legbefolyásosabb webhely a „filmek” témában** A táblázat oszlopaiban a webhely neve, az általa detektált további kaszkádok száma, és a keresőgép által hozzá rendelt relevancia százalék szerepel. A 2.1. táblázattal összevetve látható, hogy a legbefolyásosabb webhelyek nem egyeznek meg a legrelevánsabbakkal. A 20 legrelevánsabb webhely csak 523 kaszkádot detektál.



3.1. ábra. **A detektált kaszkádok aránya az összes kaszkádból** Látható, hogy az algoritmus által kiválasztott webhelyek számának növekedésével (x tengely) a detektált kaszkádok aránya először jelentősen növekszik, majd a növekedés üteme csökken. Nagyon kevés (20–40) webhely lefedi a kaszkádok nagyon nagy részét (80–90 százalékát).

3.5. Összefoglalás

Ebben a fejezetben az egy témában befolyással rendelkező webhelyek felderítését oldottuk meg egy szubmoduláris mohó gráfalgoritmus segítségével. A bemeneti gráf az előző fejezetben bemutatott keresőrobot által feltérképezett weblap–hiperhivatkozás gráf volt. Azt találtuk, hogy már kevés, 20–40 webhely figyelésével is felderíthető a hírek 80–90%-a. Ez különösen előnyös a portálkészítés szempontjából, hiszen ennyi webhely a felhasználók számára is kezelhető, illetve a friss hírek zömének detektálásához elég ezeket a webhelyeket folyamatosan figyelni.

4. fejezet

Dokumentumok jellemzése kulcsszavakkal

4.1. Bevezetés

A hírek közötti válogatást jelentősen meggyorsítja, ha rendelkezésre áll a híreknek egy tömörebb, átlátható reprezentációja. Ez egyrészt megkönnyíti a híreket szelektáló portálkészítő, másrészt a hírek között válogató felhasználó dolgát.

Dokumentumok tömör reprezentálásának egy bevált módja a kulcsszavakkal való jellemzés, ahol a dokumentumhoz a tartalmára utaló szavakat, úgynevezett kulcsszavakat rendelünk. A kulcsszavak segítségével egy pillantással megállapíthatjuk a dokumentum témáját, és következtethetünk tartalmára is.

Egy automatikus kulcsszókivonatoló ezeket a kulcsszavakat a dokumentum szövegének szavai közül próbálja meg automatikusan meghatározni, például gépi tanulás segítségével. Minden szóhoz különböző jellemzőket rendelnek, és ezeken betanítanak egy osztályozót, aminek segítségével meg lehet

becsülni az egyes dokumentumbeli szavakról, hogy mekkora valószínűséggel kulcsszavak. Ezek alapján a valószínűségek vagy pontszámok alapján ki lehet választani néhány kulcsszót, például a valószínűségek csökkenő sorrendbe rendezése után az első néhány szót.

A Keyphrase Extraction Algorithm (KEA)[1] [38], az egyik legmeghatározóbb algoritmus a területen, a következő jellemzőket használja:

- *TFIDF (Term Frequency Inverse Document Frequency)*: Az információlekérés[2] területén gyakran alkalmazott mérőszám, ami azt méri, hogy egy szó mennyire jellemző az adott dokumentumra. Egy szó akkor jellemző egy dokumentumra, ha benne sokszor, más dokumentumokban viszont kevésszer szerepel: a TFIDF mértékben a szó dokumentumbeli relatív gyakoriságát osztjuk el azon más dokumentumok számának logaritmusával, amikben még előfordul ez a szó.

- *Első előfordulás*: Azon szavak száma, amik a kérdéses szót megelőzik a dokumentumban. Azon a hipotézisen alapul, hogy a kulcsszavak előbb szerepelnek egy dokumentumban, mint a nem kulcsszavak általában.

A KEA jellemzőinek ismeretében úgy gondoljuk, hogy a weben található rövidebb szövegekből (fél, maximum egy-két oldal) nem képes olyan hatékonyan kulcsszavakat kivonatolni, mint hosszabbakból (10 oldal felett), amik egy korpuszban vannak. Egyrészt a *TFIDF* mérték egyes dokumentumokon nem, csak egy dokumentum gyűjteményen, korpuszon alkalmazható, és sok, hosszabb dokumentumot igényel a statisztikai hibák elkerülése végett. Az *első előfordulás* jellemző rövidebb dokumentumokon szintén nem alkalmazható.

[1] `http://www.nzdl.org/Kea/`
[2] angolul: information retrieval

Egy lehetséges megoldás, ha minél több információ alapján próbálunk meg dolgozni: a rövid dokumentumot feldolgozzuk, és így olyan információkkal gazdagítjuk a reprezentációját, amik segítik megtalálni a kulcsszavakat. Ennek érdekében el fogjuk végezni a szöveg gépi mondattani elemzését, ennek alapján pedig egy gráfreprezentációt készítünk, amiben a csúcsok a szavak, az élek pedig a köztük lévő mondattani relációk.

Ebben a gráfban azok a kulcsszavak, amiket környezetük a legjobban támogat. Tehát ha egy szó szomszédai magas pontszámmal rendelkeznek, megnöveljük a szó pontszámát. Például tekintsünk egy, a „mesterséges neuronhálók" témába tartozó, és ezt a kifejezést tartalmazó dokumentumot. Ha a dokumentumban sok, a témába tartozó kifejezést említenek, akkor a „mesterséges" és a „neuronhálók" szó is nagy valószínűséggel kulcsszónak fog minősülni, akkor is, ha nem túl gyakran fordulnak elő. Ugyanis a témába tartozó egyéb szavak meg fogják növelni a pontszámukat. Ha viszont egy gyakran előforduló, de a témához nem kapcsolódó szót tekintünk, mint a „tehát", az nem fog magas pontszámot kapni, mert nem támogatja a környezete. A kérdés az, hogy hogyan konkretizáljuk a támogatás fogalmát.

Egy másik, nagyon hasonló problémát old meg a PageRank algoritmus [29], ami a weblapokhoz rendel pontszámokat az alapján, hogy a környezetük mennyire támogatja őket. A pontszámok a weblap fontosságát jelképezik az összes vizsgált weblap halmazán belül. A PageRank algoritmus szerint a weboldalak készítői olyan weblapokra linkelnek, amiket jónak tartanak, így minden bemenő link felfogható egy szavazatként az adott weblapra. Egy weboldal annál fontosabb, minél több szavazatot kap, viszont a szavazatoknak is van fontosságuk, az őket leadó weboldal fontossága. Ez egy rekurzív definíció.

A definíciót mi a web helyett a dokumentumból előállított gráfra alkal-

mazzuk. A weblapokat a szavak, a linkeket a mondattani relációk váltják fel. A PageRank algoritmus a következő iterációban valósul meg:

$$\mathbf{w}^{t+1} := \alpha G\mathbf{w}^t + \big(\alpha\mathbf{a}^{\mathbf{T}}\mathbf{w}^{\mathbf{t}} + (1-\alpha)\big)\mathbf{v} \ , \tag{4.1.1}$$

, ahol a G a gráf szomszédsági mátrixából képzett sztochasztikus mátrix, a $\mathbf{w}$ vektor a szavak súlyát tartalmazza, az $\mathbf{a}$ vektor pontosan azon szavaknál tartalmaz 1-et, amelyekből nincs kimenő él, a többi eleme nulla. A $\mathbf{v}$ vektor a PageRank mátrix sztochasztikusságához kell: ha egy csúcsból nem megy ki él, a mátrix ehhez a csúcshoz tartozó sora nulla lenne. Ehelyett ezt a sort v elemeivel töltjük fel. Esetünkben v minden eleme megegyezik $\frac{1}{n}$-nel, ahol az n G dimenziója. Az $\alpha = 0,85$ paraméter ahhoz kell, hogy a PageRank algoritmus mátrixa irreducibilis legyen, az algoritmus ugyanis csak ebben az esetben konvergál. Az algoritmust csupa 1-est tartalmazó w vektorból indítjuk, tehát az egyes szavak dokumentumbeli gyakoriságát nem vesszük figyelembe. A PageRank algoritmussal részletesebben [19] foglalkozik.

4.2. A dokumentumok gráfreprezentációjának előállítása

A PageRank algoritmus alkalmas lehet a legfontosabb szavak, a kulcsszavak kiemelésére a dokumentumból. A jó megoldás kulcsa a megfelelő gráfreprezentáció megtalálása. Ehhez először előállítunk egy *kiindulási reprezentációt*, majd ezt különféle gráfátalakító modulokkal átalakítjuk, és megnézzük, hogy hogyan változik a PageRank algoritmus eredményének jósága.

A kiindulási reprezentációt három lépésben állítjuk elő. Először tokenizáljuk a dokumentum szövegét és minden szóhoz automatikusan hozzárendeljük

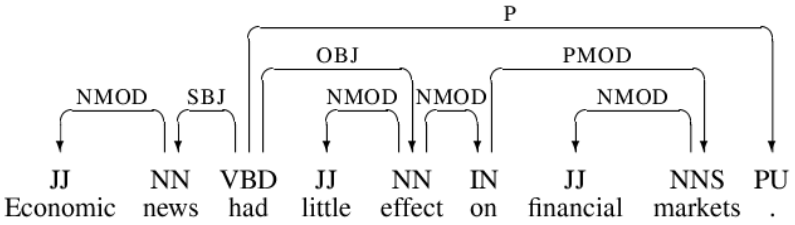
a szófaját a *Stanford Log-Linear Part-of-Speech Tagger* segítségével[3]. Ezt a a nyílt forráskódú, adatvezérelt mondattani elemzőnk, a *MaltParser*[4] [28] várja bemenetként. A MaltParser végighalad a szövegen, és minden mondatnak előállítja a mondattani szerkezetét az ún. *függőségi nyelvtan*[5] szerint (4.1. ábra).

A függőségi nyelvtan a mondatot egy függőségi fával[6] reprezentálja, ami legjobban talán a magyar nyelvű mondatok történeti mondattani elemzésének eredményéhez hasonlít. A fa elemei függőségi relációk[7]. Egy reláció egy *fej*[8] és egy *függő*[9] közti kapcsolat. A fa gyökerében rendszerint a mondat állítmánya áll. Különböző típusú relációk vannak, néhány példa:

- AMOD: melléknév vagy határozószó módosító, a függő módosítja a fej jelentését
- NMOD: főnév módosító, a függő módosítja a fej jelentését
- OBJ: tárgy, a függő a fej tárgya
- ROOT: a függőségi fa gyökerét jelöljük így
- SBJ: alany, a függő a fej alanya
- VMOD: ige módosító, a függő módosítja a fej jelentését

A függőségi relációk szemantikus tartalommal is bírnak. Például a „piros alma” szókapcsolatban egy NMOD függőségi reláció van, a „piros” melléknév módosítja az „alma” főnév jelentését. A „piros” szó az „alma” színét jelöli. A

[3] `http://nlp.stanford.edu/software/tagger.shtml`
[4] `http://maltparser.org/`
[5] angolul: dependency grammar
[6] angolul: dependency tree
[7] angolul: dependency relation
[8] angolul: head
[9] angolul: dependent



4.1. ábra. **Egy mondat függőségi fája** Az ábrán az „Economic news had little effects on financial markets." mondat függőségi fája látható. A fa gyökere a „had", a mondat állítmánya. Ehhez kapcsolódik függőként a tárgy és az alany, amikhez módosítókként jelzők és határozók kapcsolódnak.

legtöbb esetben a függő módosítja vagy pontosítja, finomítja a fej jelentését. Emiatt a függőségi relációkból készített gráf alkalmas lehet a dokumentumok jelentésének reprezentálására. A kulcsszavakat kiválasztása a PageRank működési elve szerint megvalósítható: minden szót egy weboldalnak, az éleket linkeknek tekintve a szavak „szavaznak" szomszédaik fontosságára, a kulcsszavak pedig a legfontosabbnak megszavazott szavak.

A mondattani elemző minden mondatból külön függőségi fát készít, ezeket az elemző futása után összesítjük egy G multigráfba, ami szavakat és köztük címkézett és súlyozott éleket tartalmaz. Ehhez végigmegyünk az összes fa összes csúcsán (a szavakon). A szót és a belőle kiinduló éleket (és végpontjaikat) típusukkal címkézve hozzáadjuk a gráfhoz. A különböző típusú relációkból két szó között különböző címkézett élek lesznek, súlyuk pedig az összesen talált ilyen címkéjű élek száma. Tehát ha egy olyan címkéjű élt adunk a gráfhoz, melynek végpontjai között már van a gráfban egy ugyanazzal a címkével rendelkező él, akkor nem teszünk új élet bele, hanem eggyel növeljük a súlyát. Azok a szavak, amelyek esetleg nem végpontjai egy élnek sem, nem kerülnek bele a gráfba.

Ezt a gráfot sokféleképpen át lehet alakítani a PageRank algoritmus

futása előtt, például normalizálhatjuk többféle normával a vektor normálás mintájára, vehetjük az élek logaritmusát a TFIDF számolását alapul véve, irányítatlan gráfot készíthetünk belőle az élek összeadásával, szorzásával, szemantikus hasonlóság számításával, esetleg minden él súlyát 1-re állíthatjuk. Ezen előfeldolgozó modulok tetszőleges kombinációját hajthatjuk egymás után sorban végre. Mivel a PageRank algoritmus viszonylag kis futási idejű, és a kombinációk száma nagyon nagy, úgy döntöttem, hogy az összes kombinációt végigpróbálom egy bizonyos elemszámig.

A következő átalakítókat használtam (a java osztálynevükkel hivatkozom rájuk):

- BinaryGraphTransmuter: minden él súlyát 1-re állítja
- GlobalNormalizeGraphTransmuter: Minden élsúlyt eloszt a gráfban található legnagyobb élsúllyal.
- LogGraphTransmuter: Minden e él súlyát $\ln(1 + w(e))$–re állítja, ahol $w(e)$ az él súlya.
- NormalizeInGraphTransmuter: Minden csúcsra a bejövő élek súlyát mint vektort normálja, hogy a hossza 1 legyen.
- NormalizeOutGraphTransmuter: Minden csúcsra a kimenő élek súlyát mint vektort normálja, hogy a hossza 1 legyen.
- StochasticizeInGraphTransmuter: Minden csúcsra lenormálja a bejövő élek súlyát, hogy összegük 1 legyen.
- StochasticizeOutGraphTransmuter: Minden csúcsra lenormálja a kimenő élek súlyát, hogy összegük 1 legyen.

- SymmetrizeByAddingGraphTransmuter: Egy irányítatlan gráfot készít a szembe menő élek súlyának összeadásával. Ha egy irányított éllel nincs szembe menő él, akkor változatlan súlyú irányítatlan él lesz belőle.

- SymmetrizeByContextInOutGraphTransmuter:
  Egy irányítatlan, szimmetrikus szó hasonlósági gráfot készít azon az elven, hogy a hasonló szavaknak hasonló a kontextusuk, a koszinusz hasonlósági mértéket használva (bővebben: [6]). A kimenő és a bejövő éleket is figyelembe veszi.

- SymmetrizeByContextOutGraphTransmuter: Egy irányítatlan, szimmetrikus szó hasonlósági gráfot készít azon az elven, hogy a hasonló szavaknak hasonló a kontextusuk, a koszinusz hasonlósági mértéket használva (bővebben: [6]). Csak a kimenő éleket veszi figyelembe.

- SymmetrizeByMaxingGraphTransmuter: Egy irányítatlan gráfot készít. Az új, irányítatlan él súlya a szembe menő élek súlyának maximuma lesz. Ha egy irányított éllel nincs szembe menő él, akkor változatlan súlyú irányítatlan él lesz belőle.

- SymmetrizeByMultiplyingGraphTransmuter: Egy irányítatlan gráfot készít a szembe menő élek súlyának összeszorzásával. Ha egy irányított éllel nincs szembe menő él, akkor változatlan súlyú irányítatlan él lesz belőle.

4.3. Eredmények

A módszer vizsgálatához szükségünk van egy olyan adatbázisra, amiben hosszabb és rövidebb cikkek és ezekhez megadott kulcsszavak találhatóak.

A Behavioral and Brain Sciences (BBS) archívuma[10] megfelelő erre a célra, ugyanis minden cikkhez tartoznak kulcsszavak és tartozik egy kivonat is, így a dokumentumokon tudjuk mérni a hosszabb szövegből, a dokumentumok kivonatán pedig a rövidebb szövegből történő kulcsszókivonatolás jóságát. Az archívumból 150 dokumentumot töltöttünk le. Eredményeimet a már említett KEA algoritmussal vetem össze.

A KEA algoritmus egy dokumentum korpuszt igényel a működéséhez, ami lehetőleg csak egy témába tartozó dokumentumokat tartalmaz. Emiatt céljaimra, az egyes weblapok online feldolgozására nem lenne alkalmas. Ennek ellenére a saját feltételei mellett futtattam, tehát a teljes, egy témába tartozó dokumentumokat tartalmazó korpuszon.

Az algoritmusok jóságát a már említett *pontosság* illetve *lefedettség* mértékkel számolom. Mind a két algoritmus annak a valószínűségét rendeli egy szóhoz, hogy az adott szó kulcsszó. Tehát egy szóhoz egy valós szám tartozik szemben az előző esettel, ahol egy weblaphoz egyetlen bináris döntés eredménye tartozott. Ezért a pontosság és a lefedettség fogalmát is általánosítani kell.

Legyen K_D a szerző által a D dokumentumhoz rendelt kulcsszavak halmaza, és jelöljük $x \in D$–vel, hogy az x szó a dokumentumban van, és $x \in K_D$–vel, hogy x kulcsszó. Legyen $0 \leq w(x) \leq 1$ az x szóhoz a kulcsszó kivonatoló algoritmus által rendelt súly. Így a pontosság és a lefedettség a következőképp általánosítható:

$$pontossag = \frac{\sum_{x \in K_D} w(x)}{\sum_{y \in D} w(y)} \tag{4.3.1}$$

$$lefedettseg = \frac{\sum_{x \in K_D} w(x)}{|K_D|} \tag{4.3.2}$$

[10] `http://www.bbsonline.org/`

, ahol a pontosság a kulcsszavakhoz rendelt súlyok és az összes súly aránya, a lefedettség pedig azt mondja meg, hogy a lehetséges legjobb súlyozásnak (amikor minden kulcsszó súlya 1) hány százalékát értük el. Az általánosítás a két osztály (pozitív, negatív) helyett valós súlyokat használ, működése az eredeti, bináris fogalmakkal analóg. Ha egy nem kulcsszónak magas súlyt adunk, a pontosság csökken és a lefedettség nő, ha pedig egy kulcsszónak alacsony súlyt adunk, a lefedettség csökken és a pontosság nő. Az F-mértéket továbbra is a pontosság és a lefedettség harmonikus átlagaként számítjuk.

Két kísérletet végeztem: egyet előfeldolgozás nélkül, és egyet az egyik legjobb F-mértéket adó előfeldolgozással. Ezt az előfeldolgozó modulok összes egy, kettő, három, négy, és öt elemszámú kombinációját végigpróbálva kaptam. Azt találtam, hogy a kombináció hosszát háromnál nagyobbra választva nem érhető el javulás. A három hosszú vagy ennél rövidebb legjobb kombinációk a 4.1. táblázatban találhatóak. Látható, hogy a PageRank algoritmus előtt a gráf irányítatlanná alakítása a legfontosabb, hiszen minden kombinációban csak ez szerepel. Az is látszik, hogy az algoritmus a teljes dokumentumokon jobb eredményt képes elérni a több használható szöveganyag segítségével.

Érdemes megjegyezni, hogy egy dokumentumhoz többféle jó kulcsszóhalmaz is rendelhető, emiatt a szerző által választott kulcsszavak mellett találhatunk más, jó kulcsszavakat is. Másrészt a szerző nem csak olyan szavakat választhat ki, amik benne vannak a cikkben, tehát az általa választott kulcsszavak egy részét szükségképpen nem találjuk meg. Ezért több cikkhez rendelt kulcsszóhalmazt is megvizsgáltam, és úgy találtam, hogy lehet belőlük következtetni a cikk tartalmára. Egy példa a 4.2. táblázatban látható, ami a NIPG csoport munkájáról megjelent, „Computer learns to out-munch

Teljes dokumentumok	
Használt előfeldolgozók	F-mérték
Előfeldolgozás nélkül	0.17
SymmetrizeByContextOut	0.2
SymmetrizeByMaxing SymmetrizeByMultiplying	0.22
SymmetrizeByMaxing SymmetrizeByMultiplying SymmetrizeByMultiplying	0.23

Kivonatok	
Használt előfeldolgozók	F-mérték
Előfeldolgozás nélkül	0.18
SymmetrizeByContextOut	0.19
SymmetrizeByContextOut SymmetrizeByAdding	0.19
SymmetrizeByAdding SymmetrizeByContextInOut SymmetrizeByMultiplying	0.2

4.1. táblázat. **A kombinatorikus gráf előfeldolgozás eredménye.** A felső táblázat a teljes dokumentumokon, az alsó a kivonatokon elért eredményeket mutatja. Az első oszlopban az alkalmazott előfeldolgozó modulok vannak alkalmazásuk sorrendjében, a másodikban a kombinációhoz tartozó F-mérték. A sorok az előfeldolgozás nélküli, majd az egy, kettő, illetve három előfeldolgozóval elért legjobb eredményt mutatják.

humans at Pac-Man"[11] című cikkhez rendelt kulcsszavakat mutatja.

Kulcsszó	Súly
human	0.99999994
ghost	0.89742565
lorincz	0.87548584
dot	0.69217384
player	0.59387445
such	0.53523105
pac-man	0.4562424
predict	0.3654597
pattern	0.36527765
possible	0.35594222
ai	0.355067

4.2. táblázat. **Egy, a csoportunk munkájáról megjelent cikkhez rendelt kulcsszavak** A cikk címe „Computer learns to out-munch humans at Pac-Man", és a egy, a Pac-Man játékot egy átlagos emberi játékos szintjén játszó mesterséges intelligenciát mutat be. Látható, hogy a kulcsszavak összessége jól jellemzi a cikk tartalmát. A *pac-man*-ben az emberi játékosnak (*human player*) pontokat (*dot*) kell összegyűjtenie, és menekülnie a szellemek (*ghost*) elől. A *lorincz* témavezetőm neve, míg a *predict*, *pattern* és az *ai* a használt módszerre vonatkoznak. Két szó van, melyek nem kapcsolódnak szorosan a cikk témájához, a *such* és a *possible*, bár a *possible* kulcsszó volta mellett lehetne érvelni.

Algoritmusunkat a KEA-val összehasonlítva a 4.3. táblázatból kiderül, hogy teljes dokumentumokon mind a nem előfeldolgozott, mind az előfeldolgozott gráfon dolgozó PageRank algoritmus rosszabbul szerepel a KEA algoritmusnál, de ne feledjük, hogy a feltételeket szándékosan úgy alakítottuk, hogy a KEA-t favorizáljuk. Azzal, hogy egy teljes, egy témáról szóló korpusszal dolgozunk, nem érvényesítjük algoritmusunk azon előnyét, hogy képes egyesével is feldolgozni különböző témáról szóló anyagokat. Az is látható, hogy az előfeldolgozás jelentősen javította a PageRank teljesítményét. A kivonatokon viszont fordul a kocka: algoritmusunk előfeldolgozás nélkül

[11] http://www.newscientist.com/article/dn13210-computer-learns-to-outmunch-humans-at-pacman.html

is jobb eredményt ad, az előfeldolgozással pedig tovább növeli előnyét. Kis adatigénye miatt a rendelkezésre álló adatok csökkenésével algoritmusunk teljesítménye kevésbé csökkent, mint a sok adatot igénylő KEA–é.

	Teljes dokumentumok			Kivonatok		
Algoritmus	Pont.	Lefed.	F-mérték	Pont.	Lefed.	F-mérték
KEA	0.193	0.325	0.242	0.154	0.190	0.170
PageRank	0.141	0.236	0.177	0.167	0.184	0.175
Előfeldolgozott PageRank	0.243	0.206	0.223	0.179	0.206	0.192

4.3. táblázat. **A kulcsszó kivonatoló algoritmusok összehasonlítása** Az oszlopok a KEA és a PageRank algoritmus által elért pontosságot, lefedettséget és F-mértéket mutatják a teljes dokumentumokon és a kivonatokon. Látható, hogy a PageRank algoritmus a teljes dokumentumokon gyengébben, a kivonatokon viszont jobban teljesít. Az előfeldolgozás egyértelműen javítja a PageRank teljesítményét.

4.4. Összefoglalás

Létrehoztunk a természetes nyelvfeldolgozás módszereit és a PageRank algoritmust felhasználva egy gráf alapú kulcsszókivonatoló módszert, ami eléri a területen alapvetőnek számító KEA algoritmus teljesítményét, sőt, rövidebb dokumentumokra felül is múlja azt. A módszer egy portálkészítést segítő rendszer hatékony komponense lehet, mivel képes egyesével feldolgozni a dokumentumokat (szemben a KEA-val). A kulcsszavak lehetőséget nyújtanak a hírek tartalmának gyors áttekintésére.

5. fejezet

Összefoglalás

Dolgozatom célja olyan gráf alapú módszerek kidolgozása volt, amikkel a portálkészítés feladata részben automatizálható, ezáltal egyrészt megkönnyíthető, másrészt a portálok minősége javítható. A feladatot három részfeladatra bontottam:

1. A világháló a portál témájával foglalkozó részének felderítése
2. A portál témájával foglalkozó, legmeghatározóbb webhelyek megtalálása
3. A portálon megjeleníthető hírek tömör jellemzése a hírek közötti válogatás megkönnyítésére

Dolgozatomban az első célt a 2., a másodikat a 3., a harmadikat pedig a 4. fejezetben oldottam meg.

A portálkészítés folyamata a következőképpen automatizálható:

1. Gyűjtünk a portál témájához kapcsolódó HTML dokumentumokat, az esetek többségében ezek már rendelkezésre állnak. Ezekkel felülírjuk a keresőrobot pozitív tanítópéldáit. Ha esetleg a negatív példák között

szerepelnének a témába tartozó dokumentumok, ezeket áthelyezzük a pozitív példák közé. Mivel a negatív példák kategóriákba vannak sorolva, ez nem nehéz. A tematikus keresőrobot ezután az általunk összeválogatott pozitív tanítópéldákhoz hasonló dokumentumokat fog keresni. A továbbiakban nincs szükség felhasználói beavatkozásra az eszközök működéséhez.

2. Elindítjuk a keresőgépet (2. fejezet), majd várunk, amíg bejárja és letölti a kívánt mennyiségű weblapot. A weboldalak letöltése opcionális.

3. Felderítjük a legbefolyásosabb webhelyeket a mohó szubmoduláris gráfalgoritmussal (3. fejezet). Ezeket közvetlenül is beilleszthetjük a portál hivatkozáslistájára, vagy folyamatosan figyelhetjük őket a legfontosabb hírek megtalálásához.

4. Egy hírhez tartozó információs kaszkád nagyságából (terjedésének mértékéből) következtethetünk a hír fontosságára. A beérkező hírek közül kivonatoljuk a kulcsszavakat (4. fejezet), ezután ezek egy táblázatban könnyen elhelyezhetők, amiből a portálra kerülő hírek kiválogathatók.

Látható, hogy a munkaigényes, monoton feladatokat, mint a fontos, ill. befolyásos webhelyek gyűjtése, az új hírek megtalálása, a hírek közötti válogatás, sikerült részben vagy teljes egészében automatizálni. Mindhárom komponens eléri, vagy meghaladja a napjainkban használt korszerű megoldások teljesítményét. Ezen felül fontos megjegyezni, hogy az információs kaszkádok száma egy webhelyen egy objektív mértéket is ad a webhely fontosságára, a szubmoduláris algoritmus optimálisságára pedig matematikai tételek vannak.

6. fejezet

Köszönetnyilvánítás

Először is szeretnék köszönetet mondani Lőrincz Andrásnak, témavezetőmnek, aki mindig pontosan annyi segítséget, biztatást és támogatást adott, amennyire szükségem volt, nem többet és nem kevesebbet. Erre nagyon kevesen képesek. Sokszor előfordult, hogy heteken keresztül minden nap, vagy minden másnap több órás megbeszéléseket tartottunk.

Azt, hogy hogyan volt képes időt szakítani erre a többi, öt–tíz projekt, pályázatírás és az adminisztráció mellett, nem tudom. Azt viszont tudom, hogy számomra nagyon hasznos volt minden szempontból a közösen eltöltött három év. Nemrég megkérdezte, hogy hogyan tudná jobban segíteni a munkámat. Sokat gondolkodtam rajta, de azóta sem jutott semmi sem az eszembe.

Köszönet illeti Bontovics Ákost, akivel több, mint egy évet dolgoztam együtt. Ez alatt az idő alatt mindenben támogatott, minden kérdésemre azonnal válaszolt, akár a saját munkáját is félretéve. Jó hangulatú, sokszor az estébe nyúló délutánokon munkálkodtunk a „trading"-en és a „herders"-en.

Gyenes Viktornak azt a rengeteg ötletet köszönöm, amiket a csak rá jellemző vehemenciával adott elő, másokat is magával ragadva. Dolgozatomban

a kulcsszókivonatolásról szóló fejezet nagy része az ő ötletei alapján készült.

Vörös Gyulával immár két éve dolgozom együtt. Azóta is töretlen lelkesedéssel és lelkiismeretesen veszi ki minden feladatból a részét, a legjobb tudása szerint. Dolgozatomban a KEA algoritmus futtatását ő végezte.

A csoport többi tagjának a nagyon hasznos csoportelőadásokért, segítőkészségükért, valamint a csoport működési feltételeinek megteremtésében nyújtott szerepvállalásukért mondok köszönetet. Szirtes Gábor folyamatosan írja a pályázatokat, Szabó Zoltán és Palotai Zsolt munkájának eredményei pedig sokszor nagy segítséget jelentettek, amikor közvetlenül vagy közvetve felhasználtuk őket a projektünkben.

Szabolcs Tündének köszönöm, hogy segít András adminisztrációs terheinek csökkentésében.

Árokszállási Tibor középiskolában a matematika tanárom volt. Szerencsésnek mondhatom magam, hogy ő tanított. Amellett, hogy megmutatta, mi is igazán a matematika, azt a lendületet is neki köszönhetem, ami sokszor segített tanulmányaim során. Leimsziederné Tóth Ildikónak azt köszönöm, hogy megszerettette velem az irodalmat és a történelmet.

Végezetül azoknak szeretném megköszönni a folyamatos támogatást és türelmet, akik mindig mellettem álltak, szüleimnek és testvéremnek.

Lillának azt a végtelen kedvességet, megértést és szeretetet köszönöm, amivel mindig körülvesz.

Irodalomjegyzék

[1] Charu C. Aggarwal, Fatima Al-Garawi, and Philip S. Yu, *Intelligent crawling on the world wide web with arbitrary predicates*, WWW '01: Proceedings of the 10th international conference on World Wide Web (New York, NY, USA), ACM, 2001, pp. 96–105.

[2] Sushil Bikhchandani, David Hirshleifer, and Ivo Welch, *A theory of fads, fashion, custom, and cultural change as informational cascades*, The Journal of Political Economy **100** (1992), no. 5, 992–1026.

[3] Soumen Chakrabarti, Martin van den Berg, and Byron Dom, *Focused crawling: a new approach to topic-specific web resource discovery*, 1999.

[4] Hsinchun Chen, Michael Chau, and Daniel Zeng, *Ci spider: a tool for competitive intelligence on the web*, Decis. Support Syst. **34** (2002), no. 1, 1–17.

[5] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page, *Efficient crawling through url ordering*, Comput. Netw. ISDN Syst. **30** (1998), no. 1-7, 161–172.

[6] James R. Curran and Marc Moens, *Improvements in automatic thesaurus extraction*, Proceedings of the ACL-02 workshop on Unsupervised

lexical acquisition (Morristown, NJ, USA), Association for Computational Linguistics, 2002, pp. 59–66.

[7] Brian D. Davison, *Topical locality in the web*, SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval (New York, NY, USA), ACM, 2000, pp. 272–279.

[8] P. M. E. De Bra and R. D. J. Post, *Information retrieval in the world-wide web: making client-based searching feasible*, Comput. Netw. ISDN Syst. **27** (1994), no. 2, 183–192.

[9] Franca Debole and Fabrizio Sebastiani, *An analysis of the relative hardness of reuters-21578 subsets: Research articles*, J. Am. Soc. Inf. Sci. Technol. **56** (2005), no. 6, 584–596.

[10] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori, *Focused crawling using context graphs*, VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 2000, pp. 527–534.

[11] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, *An analysis of approximations for maximizing submodular set functions.*, Math. Programming Stud. (1978), no. 14. MR MR510369

[12] William Hersh, Chris Buckley, T. J. Leone, and David Hickam, *Ohsumed: an interactive retrieval evaluation and new large test collection for research*, SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information

retrieval (New York, NY, USA), Springer-Verlag New York, Inc., 1994, pp. 192–201.

[13] Michael Hersovici, Michal Jacovi, Yoelle S. Maarek, Dan Pelleg, Menanchem Shtalhaim, and Sigalit Ur, *The shark-search algorithm. an application: tailored web site mapping*, WWW7: Proceedings of the seventh international conference on World Wide Web 7 (Amsterdam, The Netherlands, The Netherlands), Elsevier Science Publishers B. V., 1998, pp. 317–326.

[14] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan, *A dual coordinate descent method for large-scale linear svm*, ICML '08: Proceedings of the 25th international conference on Machine learning (New York, NY, USA), ACM, 2008, pp. 408–415.

[15] Thorsten Joachims, Fachbereich Informatik, Fachbereich Informatik, Fachbereich Informatik, Fachbereich Informatik, and Lehrstuhl Viii, *Text categorization with support vector machines: Learning with many relevant features*, 1997.

[16] Judy Johnson, Kostas Tsioutsiouliklis, and C. Lee Giles, *Evolving strategies for focused web crawling*, ICML, 2003, pp. 298–305.

[17] David Kempe, Jon Kleinberg, and Éva Tardos, *Maximizing the spread of influence through a social network*, KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (New York, NY, USA), ACM, 2003, pp. 137–146.

[18] Samir Khuller, Anna Moss, and Joseph (Seffi) Naor, *The budgeted maximum coverage problem*, Inf. Process. Lett. **70** (1999), no. 1, 39–45.

[19] Amy N. Langville and Carl D. Meyer, *Deeper inside pagerank*, Internet Mathematics **1** (2004), no. 3, 335–380.

[20] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance, *Cost-effective outbreak detection in networks*, KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (New York, NY, USA), ACM, 2007, pp. 420–429.

[21] Dirk Lewandowski, *A three-year study on the freshness of web search engine databases*, J. Inf. Sci. **34** (2008), no. 6, 817–831.

[22] A. Lőrincz, I. Kókai, and A. Meretei, *Intelligent high-performance crawlers used to reveal topic-specific structure of the WWW*, International Journal of Foundations of Computer Science **13** (2002), 477–495.

[23] A. Lőrincz, K. A. Lázár, and Zs. Palotai, *Efficiency of goal-oriented communicating agents in different graph topologies: A study with internet crawlers*, Physica A **378** (2007), 127–134.

[24] Filippo Menczer and Richard K. Belew, *Adaptive retrieval agents: Internalizing local contextand scaling up to the web*, Mach. Learn. **39** (2000), no. 2-3, 203–242.

[25] Filippo Menczer, Gautam Pant, and Padmini Srinivasan, *Topic-driven crawlers: Machine learning issues*, ACM TOIT, Submitted **2002** (2002), http://dollar.biz.ui.

[26] Filippo Menczer, Gautam Pant, Padmini Srinivasan, and Miguel E. Ruiz, *Evaluating topic-driven web crawlers*, SIGIR '01: Proceedings of the

24th annual international ACM SIGIR conference on Research and development in information retrieval (New York, NY, USA), ACM, 2001, pp. 241–249.

[27] Gordon Mohr, Michael Stack, Igor Ranitovic, Dan Avery, and Michele Kimpton, *An introduction to heritrix*, 4th International Web Archiving Workshop, 2004.

[28] J. Nivre, J. Hall, and J. Nilsson, *Maltparser: A data-driven parser-generator for dependency parsing*, Proceedings of the fifth international conference on Language Resources and Evaluation (LREC) (Genoa, Italy), 2006, pp. 2216–2219.

[29] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, *The pagerank citation ranking: Bringing order to the web*, 1999.

[30] Zs. Palotai, S. Mandusitz, and A. Lőrincz, *Distributed novel news mining from the internet with an evolutionary news forager community*, Int. Joint Conf. on Neural Networks 2004, 2004, IEEE Catalog Number: 04CH37541C, Paper No. 1095.

[31] G. Pant, P. Srinivasan, and F. Menczer, *Crawling the web*, 2004.

[32] Gautam Pant and Filippo Menczer, *Topical crawling for business intelligence*, ECDL, 2003, pp. 233–244.

[33] Gautam Pant and Padmini Srinivasan, *Learning to crawl: Comparing classification schemes*, ACM Trans. Inf. Syst. **23** (2005), no. 4, 430–462.

[34] M. F. Porter, *An algorithm for suffix stripping*, (1997), 313–316.

[35] Jason Rennie and Andrew Kachites McCallum, *Using reinforcement learning to spider the web efficiently*, Proceedings of the 16th International Conference on Machine Learning (ICML), 1999, pp. 335–343.

[36] P. Srinivasan, F. Menczer, and G. Pant, *A general evaluation framework for topical crawlers*, Inf. Retr. **8** (2005), no. 3, 417–447.

[37] Vladimir N. Vapnik, *The nature of statistical learning theory (information science and statistics)*, Springer, November 1999.

[38] Ian H. Witten, Gordon W. Paynter, Eibe Frank, Carl Gutwin, and Craig G. Nevill-Manning, *Kea: practical automatic keyphrase extraction*, DL '99: Proceedings of the fourth ACM conference on Digital libraries (New York, NY, USA), ACM, 1999, pp. 254–255.