Gráfos rangsoroló algoritmusok a World Wide Web keresőiben

Jakabfy Tamás
jaksy@cs.elte.hu

Diplomamunka,

Eötvös Loránd Tudományegyetem, Természettudományi Kar, alkalmazott matematikus szak

Témavezető: ifj. Benczúr András, tanársegéd, Eötvös Loránd Tudományegyetem, Természettudományi Kar, Operációkutatás Tanszék

Budapest, 2002. június

Tartalomjegyzék

Kivonat

A dolgozat a Wold Wide Webből vett mintából való információkinyerés módjaival foglalkozik, amelynek célja egy jól működő kereső létrehozása a Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutatóintézetében.

A dolgozat röviden áttekinti az Internet, és ezen belül különös tekintettel a World Wide Web kialakulását és struktúráját. A feladatnak megfelelően ezután az információkinyerési formák, lehetőségek bemutatása következik, majd az irreleváns információ kiszűrése a következő téma, különös tekintettel a megtévesztési módszerek beazonosítására. Ennek fényében több, a web linkstruktúráját felhasználó rangsoroló algoritmus kerül bemutatásra. Az algoritmusok leírása és rövid elemzése után a web felépülésének egy általános modellje következik, majd a már megismert algoritmusok negatív és pozitív tulajdonságaira mutat rá a tesztelésükkel foglalkozó fejezet. Legvégül az algoritmusok stabilitásának elemzése kerül sorra.

A dolgozat elkészítésében segítségemre voltak, ezért köszönettel tartozom:

Benczúr Andrásnak, témavezetőmnek, aki folyamatosan tanácsokal segített, és a dolgozatot többször átnézte,

Grolmusz Vincének, aki a témával először megismertetett,

Csalogány Károly programozó matematikus hallgatónak, aki a programozási kérdésekben, a gráfstruktúra adatszerkezeti kezelésében és a *PageRank* algoritmus témakörében segített sokat,

Uher Máté évfolyamtársamnak, aki a web keresők architektúrájával foglalkozik,

Windhager Eszter évfolyamtársamnak, aki az adatbázis frissítési stratégiáin dolgozik,

Sarlós Tamásnak, aki az indexelő modult készítette el, valamint

Fogaras Dánielnek, aki az SVD-n alapuló algoritmusok témájában volt segítségemre.

1. Bevezetés

Az *Internet* hálózatában jelenleg is egy exponenciális növekedés tapasztalható a legtöbb mérhető dimenzióban (felhasználók száma, a tartalom adatmennyisége, [3], [14]). Ez az új technológiáknál általában így zajlik, mint azt a XX. század technikai vívmányainál is láthattuk már, a helyzet kísértetiesen hasonlít a rádió, a televízió, a telefon és a személyi számítógép elterjedéséhez. Az Internetre kapcsolt eszközök manapság számítógépeket jelentenek, azonban a jövőben sok más eszköz Internetre kapcsolását tervezik, így várhatóan az Internet nagyságában meglevő exponenciális növekedés sokkal tovább lesz fenntartható, mint a számítógépek számának emelkedése. Az Internetre kötött eszközöket a továbbiakban csak *számítógép*eknek fogjuk nevezni.

Az Internet egy teljesen *decentralizált* hálózatot jelent, amely számítógépek és a köztük levő adatátviteli vonalak halmazaként definiálható. Két számítógép kapcsolatot tud teremteni egymással, ha van köztük – esetleg más számítógépeken keresztül vezető – *adatátviteli útvonal*, és létezik egy "közös nyelv", amelyeket ezesetben *protokoll*oknak neveznek. Az Internet alapprotokolljai a TCP (Transmission Control Protocol) és az IP (Internet Protocol). Ezek definiálják a számítógépek kapcsolattartásának módjait, az egymásnak küldött adatcsomagok formátumát, a hibajavítás módját, a hosszú üzenetek darabolását, majd újra összerakását, az alapszintű forgalomszabályozást. Ezek felett helyezkednek el a magasabb szintű protokollok, amelyek a valóságban használt szolgáltatásokat írják le (*ftp, smtp, ssh, telnet, http*, stb.). A hálózat kliens-szerver modell alapján működik, a kliensprogram létrehoz egy kérést, majd továbbítja a TCP/IP protokolloknak, amelyek elvégzik a kérés csomagokra bontását, majd a csomagok átküldését a hálózaton, később a szerveroldalon a kérés újbóli összerakását és átadását a szerverprogramnak. A szerver által küldött válaszüzenettel ugyanez történik fordított sorrendben.

Ebben a munkában csak a ***HTTP***-t (*Hypertext Transfer Protocol*) használó kommunikációt vizsgáljuk. Ez a protokol a 90-es évek elején született, több, mint 20 évvel az Internet születése után. Viszont jelenleg a hálózati forgalomnak elsöprő többsége (több, mint 90%) zajlik közvetítésével, a fejlődés exponenciális dinamikája is ennek köszönhető. A protokolt egyszerű szöveges dokumentumok lekérésére és kiszolgálására hozták létre, amelyeknek azonban volt egy új tulajdonságuk: definiálhatók voltak ún. *link*ek egy dokumentumról egy másikra, amelyekre kattintva a másik dokumentum letöltődött. Nem kellett tehát mindig tudni, hogy melyik kiszolgálón milyen anyagok vannak, elég volt, ha csak egyvalaki tudta ezt előttünk.

A ***HTML*** (*Hypertext Markup Language*) a HTTP segítségével továbbított dokumentumok leíró nyelve. Tartalmazza az oldalt formázó, a megjelenítését szabályozó elemeket, valamint a dokumentumhoz csatolódó egyéb egységek (képek, egyéb multimédiás anyagok, futtatható programok, linkek, stb.) kódját vagy elérési útját, illetve több meg nem jelenített információt (továbbiakban *metaadatok*, például ilyen lehet a szerző neve, a kulcsszavak stb.). Az összes HTML dokumentumok halmazát nevezzük még *World Wide Web*nek (*WWW, web*) is. A dokumentumok helyét az Interneten egy olyan ún. *URL* (*Uniform*

Resource Locator) adja meg, amelyben a protokoll helyén *http://* áll. Egy ilyen URL tartalmazza a gazda számítógép nevét, és dokumentum elérési útját az ottani könyvtárszerkezetben. Tartalmazhat továbbá extra információkat is (pl. jelszó, és felhasználói név az olvasási jogosultság korlátozására), az egyszerűség kedvéért ezekkel a továbbiakban nem foglalkozunk.

A HTTP kliensprogramjait gyűjtőnéven ***böngésző***knek nevezzük. Ezek a böngészők jelenítik meg a szerverről letöltött forrás dokumentumformázási parancsai szerint az oldalt. Jelenleg már nemcsak szöveges információ érhető el a böngészőkkel, legtöbbjük képes multimédiás anyagok (képek, hangok, animációk) megjelenítésére, valamint programok futtatására is.

A WWW a fentiek szerint egy ***gráfstruktúrá***val írható le, és mi a továbbiakban az egész dolgozatban ezt a modellt használjuk: a *gráf* csúcsaiban a dokumentumok találhatók, míg egy dokumentumból egy másikra mutató link egy élet határoz meg a gráfban. (A továbbiakban a *dokumentum*, *oldal*, *honlap* és *csúcs*, illetve a *link* és az *él* szavakat szinonimaként használjuk.)

2. Keresés az Interneten

Az Internet (pontosabban a WWW) növekedésében kulcsszerepet játszottak a keresőrendszerek. Ezeknek használatával sokkal könnyebb volt a keresett témájú oldalakat megtalálni, mint korábban, hiszen elég volt néhány kulcsszót beírni, és máris visszaadott a kereső néhány (vagy rengeteg) oldalt, amely a megadott kulcsszavakhoz kapcsolódik. Kísérleteztek az oldalak katalógusba rendezésével is, de ez a robbanásszerű növekedés eljövetelével egyre jobban háttérbe szoruló technika lett, hiszen az oldalak számával arányos élőmunkát kívánt meg, szükségessé vált tehát az összes keresőrendszer végezte munkának az automatizálása.

Elkezdték működésüket tehát az ún. ***keresőrobotok***, amelyek folyamatosan oldalakat töltöttek le az Internetről, és egy adatbázisban tárolták ezeket. Ebben az adatbázisban aztán kereséseket (lekérdezéseket) lehetett indítani. Azonban minden témában sok ezer, esetleg sok millió oldal közül kellett megtalálni a legjobb minőségű, a témához legjobban illeszkedő tartalmakat, így ezen a ponton sok kérdés adódott hirtelen az új technológiát bevezetni kívánó vállalatnál:

- Milyen algoritmust használjunk a lekérdezésre adandó válasz dokumentumhalmaz meghatározásánál?
- Milyen sorrendben jelenítsük meg ezeket az dokumentumokat? Ennek a sorrendnek a meghatározásához milyen információkat használjunk, ezeket milyen korlátok közt és milyen súlyozással?
- A folyamatosan változó dokumentumok tartalmát milyen stratégia szerint frissítsük?
- Milyen módszerrel próbáljunk az újonnan keletkezett dokumentumok nyomára akadni?

Jelen dolgozat főleg a második kérdéskörrel foglalkozik. A következő fejezetben megvizsgáljuk, milyen adatokból indulhatunk ki, ha az oldalak egy (fontossági) sorrendjét szeretnénk meghatározni. A dolgozat további részei az *adatbányászati módszerek*et írják le, az adatszűréstől kezdve a strukturális modellezésen át a konkrét rangsoroló algoritmusokig. A dolgozat végén ezek működését vizsgáljuk mesterségesen generált (valamilyen extrém helyzetet vagy esetleges csalási módszert modellező) példákon, valamint a magyar WWW-ből vett mintán is teszteljük ezeket. Szót ejtünk még az algoritmusok stabilitásáról is.

Ebben a fejezetben a többi felvetett kérdésre olyan módon próbálunk válaszolni, hogy a *Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutatóintézet*ében fejlesztés alatt álló WWW-kereső részeit, architektúráját röviden bemutatjuk. (Bővebben lásd: [3], [19])

A kereső adatbázisába egy **robot** gyűjti a dokumentumkat. A robotot egy jó linkgyűjteményről (jelen esetben *www.cs.elte.hu*) indítjuk, és ettől az oldaltól kezdve lényegében szélességi kereséssel járja be a webgráfot (azaz itt a *.hu* végződésű oldalakat). A robot ezután két adatbázist menedzsel. Az egyik adatbázisban a már letöltött oldalak vannak, a másikban pedig azok az URL-ek,

amelyek valamely már letöltött oldalban előfordulnak, de még nincsenek letöltve (technikailag más adatbázisok is készülnek, például azokból az URL-ekből, amelyeket már megpróbált letölteni, de hibaüzenetet kapott, vagy időtúllépés miatt nem sikerült a letöltés). Amennyiben egy dokumentumot letöltött a robot, akkor az URL-jét átteszi a le nem töltöttek listájából a letöltöttek listájába, és az összes linket kiveszi a dokumentumból, és az URL-jeiket beteszi az ismert, de le nem töltött oldalak listájába (amennyiben eddig még nem voltak bent). A robot tehát *linkeket követve* térképezi fel a WWW-t.

Az így letöltött oldalakat aztán az **indexelő** veszi kezelésbe, amely a dokumentumokban előforduló szavakat nyeri ki, és írja egy újabb adatbázisba. A továbbiakban ha egy szóra keresünk, akkor ez az index fogja megmondani, mely dokumentumok esélyesek egyáltalán arra, hogy a találati listába kerüljenek. Az indexelő általában nemcsak szöveges, hanem struktúra és hasznossági indexet is készít.

A következő modul a **rangsorolás**é, ennek leírása képezi a dolgozat gerincét, most ezt nem részletezzük.

Nagyon fontos rész a **frissítés** ([12]), a már letöltött és indexelt oldalak újra letöltése, és ezzel az adatbázis folyamatos frissen tartása. Az ezirányú stratégiák sok kompromisszumot igényelnek, lévén a nagyon gyakran változó oldalakat nagyon nehéz frissen tartani, viszon a ritkán vagy soha nem változóakat felesleges gyakran letöltögetni. Ennek a résznek szintén nagyon mély matematikai tartalma van.

Az utolsó rész a **keresőfelület**, amely a keresés rangsorolt találataiból kivonatot készít, és egy HTML oldalt generál belőlük.

3. Az webből való információkinyerés formái

A web napjainkra olyan óriási méretet ért el, hogy még az emberek elenyésző részét érdeklő témákban is legalább több ezer, a népszerűbb témákban pedig több millió oldal található. Ezekből kell egy keresőnek a legjobbakat kiszűrnie, így a hagyományos technikáknál sokkal automatizáltabb, ugyanakkor nagy biztonsággal működő információkinyerő modul szükséges.

A dokumentumok rangsorolásának elvégzéséhez szerteágazó algoritmuscsaládok használatosak. Legfőképpen a *dokumentumok szavai* fontosak, hiszen egy keresést úgy adunk meg, hogy néhány *kulcsszó*t jelölünk ki. Egy olyan dokumentum, amely tartalmazza ezeket a szavakat, sokkal relevánsabb a keresés szempontjából, mint az olyan oldalak, amelyek ezeket nem tartalmazzák. Mindenképp érdemes használni a böngészőkben meg nem jelenő dokumentumelemek közül az oldal *kulcsszavai*t, *leírás*át.

Az egyik legfontosabb mértéke annak, hogy a külvilág hogy vélekedik egy adott oldal fontosságáról, a web *linkstruktúrá*jából nyerhető ki. Ugyanis ha valakinek tetszik egy oldal, elhelyez egy linket rá a saját honlapjáról. Információval szolgálhatna még, ha tudnánk az oldalak *népszerűségi rangsor*át, de ezen adatok auditálását csak a nagy oldalak engedhetik meg maguknak. Tudunk viszont támaszkodni a *kulcsszavak népszerűségi rangsorá*ra, amelyet a kereső hosszú idejű működése után a *keresett szavak és kifejezések statisztikái*ból nyerhetünk.

Itt kell megemlítenünk azt a tényt, hogy a web esetében a hagyományos információkinyerési technikák csak erősen *korlátozva alkalmazhatók*. A web ugyanis egyrészt roppant *nagy méretű*, nagyon *gyorsan növekszik* valamint nagyon *dinamikus*, egyes források szerint ([3]) az oldalak közel egy negyede naponta változik. Tehát az adatbázis, amelyből az *információkinyerő algoritmusok* dolgoznak, meg sem közelíti a dokumentumok tartalmának valódi állapotát. Ennél nagyobb *stabilitás*t mutat a *linkstruktúra*, különösen a „külső" oldalakra mutató linkek (lásd a 4.2. fejezetben) struktúrája. Fontos továbbá megemlíteni, hogy a web nyílt (*szabványosított*) szerkezete miatt bárki tud weboldalt készíteni, így a keletkező hálózat nagyon *heterogén* lesz, az oldalak minősége és célja nagyon széles skálán változik. Ebből a csak részben ismert, gyorsan változó, nagy, kaotikus rendszerből kell a lehető legbiztosabb eredményt adnia az információkinyerő algoritmusainknak.

De nem is az előzőek miatt a legrosszabb a helyzet. Az oldalak szerzői, a *tartalomszolgáltatók* ugyanis az első keresők megjelenésével felismerték, hogy óriási forgalmat tudnak az oldalaikra terelni, ha a keresők algoritmusait meg tudják téveszteni. Innentől kezdve tehát minden minden szolgáltatónak elemi érdeke lett az oldalak forráskódjába manipulatív elemek elhelyezése, így mára a keresők üzemeltetői számára ezeknek a *manipulációk*nak a kiszűrése vált elsőrendű feladatává.

A következő alfejezetekben összefoglaljuk azokat a dokumentumrészleteket és a hozzájuk kapcsolódó technikákat, amelyeket eredményesen lehet alkalmazni az információnyerés folyamatában.

3.1. A dokumentumok szavai, szókapcsolatok

A dokumentumokból legegyszerűbben a **szöveges információ** nyerhető ki, így a rangsorolás alapja ez lehet. Azt, hogy egy dokumentum milyen *téma*ról tartalmaz információkat, nagy bizonyossággal megállapíthatjuk a benne előforduló szavakból. Felvehető ugyanis egy *szó-dokumentum mátrix*, amely az i. szó és a j. dokumentum helyén azt a számot tartalmazza, ahányszor az i. szó a j. dokumentumban előfordul (illetve – ha az előfordulása fontos – 0-t vagy 1-et aszerint, hogy tartalmazza-e a dokumentum a szót). Ennek a mátrixnak sajnos mindkét dimenziója kezelhetetlen nagyságú lehet, így explicit felírása nem javasolt, viszont ritka mátrix, így éllistában jól tárolható. A mátrixos modellben azonban minden dokumentumot egy *vektor* jellemez, amely vektorok között valamilyen **normá**t használva *távolságfogalmak*at definiálhatunk. Jó eséllyel azonos témával foglalkozó oldalak kerülnek egy csoportba, amennyiben erre építve egy *klaszterező algoritmus*t ([13]) futtatunk le. Egy ilyen algoritmus lehet építő vagy osztó jellegű. Egy építő algoritmusnak meg kell adni előre pontokat, ezeket beteszi egy-egy halmazba, és az összes további pontot ezen halmazok valamelyikébe teszi az aktuális halmazoktól való távolság alapján. Az osztó algoritmus az összes pontot egy halmaznak tekinti, majd kettéosztja, a továbbiakban pedig mindig a legnagyobb halmazt osztja két részre a leállási kritérium eléréséig.

A dokumentumok szavait használó algoritmusok hátránya, hogy a természetes nyelvben egy szó és a jelentése között nem állítható fel egy-egyértelmű kapcsolat. Lehet egy szó *több jelentésű*, ilyenkor azon bosszankodhatunk, hogy a kereső a beírt szó egy másik jelentését tartja fontosabbnak, így csak ilyen témájú találatokat ad ki. Vannak továbbá olyan szavak, amelyek ugyanazt jelentik, így – ha csak az egyikre keresünk – sok információ rejtve maradhat előttünk, amely a keresésünk szempontjából nagyon releváns.

3.2. Metaadatok

Ezek az adatok nem jelennek meg a böngésző ablakában, viszont a *dokumentum forrása* tartalmazza ezeket. Ilyenek a szerző által megadott **kulcsszavak**, az oldal **leírás**a stb. Általánosságban elmondható, hogy ezen adatok kezelésére ugyanazok a szabályok és figyelmeztetések vonatkoznak, mint amelyeket az előző részben leírtunk, hiszen ezek is szöveges adatok. A kulcsszavak viszont segíthetnek jobban behatárolni a dokumentum témáját. Például amennyiben egy oldal kulcsszavai között szerepel a *puma* és a *szabadidősport* szó, akkor ennek az oldalnak a súlyát csökkenteni kell a „*puma és szavanna*” keresés esetén, mivel valószínűsíthetően teljesen más a témája az oldalnak és a keresésnek. Ugyanígy annak az oldalnak a súlyát növelni érdemes, amelynek címében, vagy kulcsszavai között szerepel a *csúcsragadozó* vagy *macskafélék* vagy *fauna* szó.

Összefoglalva a metaadatok az oldal témájának meghatározásában lehetnek segítségünkre, ehhez viszont el kell készítenünk először egy olyan adatbázist, amely témák szerint tartalmazza az ahhoz a témához tartozó releváns szavakat. Az ilyen adatbázis elkészítése nehéz feladat, hiszen például magának a témának a definíciója sem adható meg precízen. Erre tesz egy kísérletet a 6. fejezetben leírt

általános modell. A jelenleg létező hasonló adatbázisok általában empirikusan, heurisztikusan készültek.

Szót kell továbbá ejtenünk arról, hogy megjelentek különleges *kiemelő elemek* az oldalakban (aláhúzás, dőlt betű, vastag betű, villogó szöveg, stb.), amelyeknek vizuális hatásuk van ugyan (tehát nem tekinthetők metaadatoknak), viszont az oldal kulcsszavai közé bátran felvehetők ([10]). Ez többször megoldja azt a problémát is, hogy a szerzők sokszor elfeledkeznek a metaadatok megadásáról, illetve szándékosan vagy ismerethiányból fakadóan nem adják meg azokat. Egy jó keresőrendszer ezesetben is megpróbálja az oldal kulcsszavait beazonosítani, ehhez egy jó kezdeti eszköz a fenti kiemelt szövekek kulcsszóként való kezelése.

3.3. A linkstruktúra

Az oldalak linkstruktúráját ***irányított gráf***fal ábrázoljuk, ahol a csúcsok a dokumentumok, és egy csúcsból egy másikba pontosan akkor megy él, ha a hozzá tartozó dokumentum tartalmaz egy linket a másik dokumentumra. Ez az irányított gráf ugyan sok élet és sok csúcsot tartalmaz, mégis szerencsére egyes mérések szerint ([6], [5], [4], [17]) elég ritka lesz. Egy linket úgy tekinthetünk, hogy az egy **szavazat**, amely annak az oldalnak a tartalmasságát ismeri el, ahova mutat. A dokumentumok szerzői tehát egymás munkáját ebben a struktúrában értékelik, egy, a linkstruktúrára épülő rangsoroló algoritmusnak ezeknek az értékeléseknek az összevetéséből kell ítélnie. Mélyebben meggondolva a linkstruktúra az oldalak tartalmi, hasznossági elemzése során áll elő, ezt végzi el a többi oldal tulajdonosa, mielőtt linkel az adott oldalra. Ez persze minden egyes link esetén *szubjektív* és *hiányos információjú döntés*, viszont ha feltételezzük, hogy ezek a negatív hatások a globális weben kiegyenlítik egymást, akkor nyugodtan építhetünk erre algoritmust. Több ilyen algoritmust ismertetünk az 5. fejezetben, a dolgozat gerinceként. Ezeknek az algoritmusoknak a tesztelése a 7. fejezetben, stabilitásvizsgálata a 8. fejezetben található. Valós webből gyűjtött mintán is alkalmaztuk az algoritmusok egy részét, de ennek eredményei még nem kellően stabilak, további teszteléseket kívánnak.

3.4. A dokumentumokban elhelyezett multimédiás anyagok

Manapság a dokumentumok lassan elvesztik szöveges jellegüket, általában egy oldalon a rajta levő szöveges információnak sokszorosa jelenik meg *multimédiás* (kép, hang, animáció, stb.) formában. Ez azt jelenti, hogy a jövő keresőjének ezeket az adatokat nem szabad figyelmen kívül hagyni. Ma sajnos nem léteznek olyan hatékony algoritmusok, amelyek például egy kép alapján egy témát tudnának kijelölni, amely a képhez köthető. Az ilyen algoritmusok készítéséhez általában mesterséges intelligencián alapuló tanulórendszereket próbálnak igénybe venni, azonban ennek a feladatnak a megoldása valószínűleg még hosszú folyamat lesz. Nem is beszélve arról, hogy egy hangeffektus általában nem, vagy nagyon szubjektíven köthető csak témához.

Addig is az ezekről a fájlokról rendelkezésre álló korlátozott információkat be tudjuk a keresőbe építeni. Ezek a file neve, és a belőle esetleg kiolvasható leírás (tulajdonképpen a file metaadatai). Használhatjuk továbbá a multimédiás anyag elhelyezkedéséhez közeli szövegrészeket is az anyag témájának eldöntéséhez, hiszen ezek az anyagok általában központi részletet jelentenek a megjelenített oldalon, amely maga után vonja előzetes vagy utólagos magyarázó szöveg meglétét.

3.5. Népszerűségi rangsor

Az imént esett szó arról, hogy amennyiben valaki a saját lapján egy másik oldalra egy linket helyez el, azt egy *szavazat*nak számítjuk. Ugyanígy természetesnek tűnik az a megközelítés, hogy egy dokumentumra leadott szavazatnak a dokumentum egy *letöltés*ét vegyük. Ezeknek az adatoknak az összesítése adja ugyanis a legjobb népszerűségi rangsorát az oldalaknak, és egy népszerűbb oldalról joggal gondolható az, hogy tartalmasabb, több hasznos információt tartalmaz, mint egy kevéssé népszerű dokumentum. Ebben a modellben figyelembe vesszük azokat az embereket is, akik használják a webet, de nincs saját oldaluk, amelyekre linkeket tudnának elhelyezni. Tulajdonképpen ezek az emberek nagyobb részét alkotják a webet használóknak, mint az aktív honlapszerkesztők.

Sajnos azonban ezek az adatok nem állnak olyan átfogóan rendelkezésre. Nagyon kevés weboldal (tulajdonképpen csak a professzionális tartalomszolgáltatók) engedheti meg magának, hogy ilyen adatokat auditáltasson. A saját bevállásra pedig nem építhetünk, hiszen eleve feltételezhető az összes weboldaltulajdonos bevezetőben említett manipulatív szándéka.

Van azonban egy olyan módszer, amelyet a kereső alkalmazhat és valamennyire tükrözi a web „közönségének" népszerűségi rangsorát. A kereső folyamatos működése során ugyanis nagy mennyiségű adat gyűlik össze a keresésekről, ezekre az adatokra épülve rangsorba állíthatók a keresett kulcsszavak valamint a keresésre adott válasz dokumentumok is. Ez a rangsor aztán használható a frissítési stratégia kialakításánál, de akár az oldalak rangsorába is beszámítható.

4. Az irreleváns információ kiszűrése

A web óriási adathalmazában rengeteg **haszontalan információ** van. Sőt rengeteg rosszindulatúan elhelyezett adat is található, amelyek segítségével a kereső rangsorolását próbálják befolyásolni, hogy a saját oldalukat tartalmasabbnak feltüntetve a keresőt használók közül a saját forgalmukat növeljék. Azonban egy jó keresőnek fel kell készülni az ilyen manipulatív céllal létrehozott információhalmaz kiszűrésére.

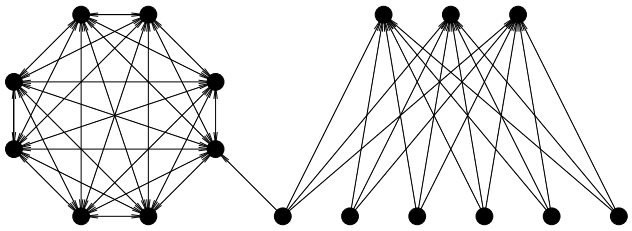
4.1. A keresők becsapásának módjai

Egyidős a keresőkkel a honlaptulajdonosoknak az a vágya, hogy a keresőkben az ő honlapjaik jelenjenek meg az első helyen. A keresés indítója ugyanis általában maximum az első tíz találat közül látogat el néhány helyre, a többit figyelmen kívül hagyja.

Szöveges és/vagy metaadatokat használó keresők esetén ez viszonylag könnyű feladat volt, hiszen tulajdonképpen korlátlan számú szót lehetett elhelyezni egy oldalon (ez különösképpen igaz a metaadatokra). Ezzel az oldal egy lokális tartalmas oldalt hozott létre, amely azokat a keresőket félrevezette, amelyek csak az oldal szavaiból ítélték meg az oldal fontosságát. Ez a technika valószínűleg örökké tovább fog élni, hiszen soha nem lesz olyan jól működő kereső, amely a dokumentumok szavait figyelmen kívül hagyná. Azonban az egyéb – főleg a linkstruktúrát használó – algoritmusok előtérbe kerülésével ez a módszer nagyságrendeket vesztett hatékonyságából.

Emiatt azonban megjelentek olyan módszerek, amelyek olyan lokális linkstruktúrát hoznak létre, melynek segítségével a rangsorolást félre lehet vezetni. Mivel ezek a módszerek költségvonzata szinte nulla, hiszen új oldalak létrehozása még új számítógépet sem igényel, ezért ezek nagyon elterjedtek lettek. Tipikus megjelenésük egy teljes részgráf létrehozása. Ez látható az 1. ábrán, amelyben a az azonos témájú oldalak által alkotott tipikus jobb oldali közösségtől (teljes páros gráf) a bal oldalon látható teljes részgráf manipulatív lokális linkstruktúrája sok rangsoroló algoritmus esetén szinte az összes pontszámot el tudja venni.

Algoritmusainknak ilyen és ehhez hasonló trükkök ellen való védettségét a 7. fejezetben tárgyaljuk, megvizsgálva azt, hogy a lokális linkstruktúra létrehozása félrevezeti-e a kereső 5. fejezetben tárgyalt algoritmusait, azaz a valódinál nagyobb fontosságot tulajdonít-e az abban szereplő oldalaknak.

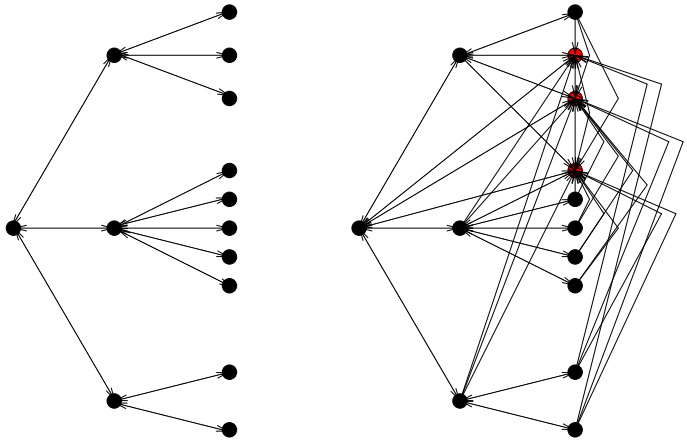


1. ábra. Szoros lokális közösség létrehozása a rangsorolás félrevezetése céljából

4.2. Egymáshoz kapcsolódó oldalak

A linkek jó része *nem informatív*. Ezt az állítást mi sem támasztja jobban alá, mint az, hogy nap, mint nap találkozunk olyan linkekkel, amelyek egy oldalrendszeren belül levő dokumentumok közötti navigációs célokat szolgál. A dokumentumokat tehát célszerűnek látszik nem önmagukban kezelni, hanem kisebb csoportokba összehúzni. Ezeket a csoportokat a továbbiakban *domain*eknek nevezzük. Azért is jó eljárás ez, mert egy domainen belüli nem navigációs link sem biztos, hogy jó információhordozó, hiszen sokszor egy portál szerkesztőjének még akkor sem szabad a konkurencia lapjára linkelni, ha esetleg ott sokkal részletesebb, szerteágazóbb információk találhatók, előírás, hogy egy helyi, esetenként sokkal szegényesebb oldalra kell linket elhelyezni.

A mi modellünkben tehát a legegyszerűbb módon az összes domainen belüli linket figyelmen kívül hagyjuk. Valószínűleg itt is – mint annyiszor – az igazság a két szélsőség között van, a domainen belüli linkek között is akadnak informatívak. Ennek megállapításához egy olyan algoritmus volna szükséges, amely megállapítja az egy domainen belüli oldalak közös struktúráját, és kiszűri a navigációs linkeket. Ez az algoritmus már szükségszerűen mesterséges intelligenciát használ, így túllép e dolgozat keretein.



2. ábra. A friss oldalakra való automatikus linkelés linkstruktúra-zavaró hatása

Általában egy ilyen struktúra a következőképpen néz ki: létezik egy főoldal, ezen kétfajta link van, egyrészt a legfontosabb újdonságokra, másrészt a különböző témájú aloldalakra mutató linkek. Ezeken az aloldalakon a témájuknak megfelelő legfontosabb friss anyagokra és az esetleges további aloldalakra mutató linkek vannak. Három szintnél mélyebb struktúra már ritkának számít. E struktúra illusztrációja a 2. ábrán is megtalálható, bal oldalon az újdonságokra mutató általában automatikusan generált linkek nélkül látható a struktúra, míg jobb oldalt ezekkel együtt.

A struktúrát egy példán is szemléltetem: egy friss híreket szolgáltató portáloldalon külön alrendszerbe vannak szervezve a politikai, kulturális, számí-

tástechnikai, gazdasági és sporthírek. A megfelelő rovatok alatt pl. belpolitika/külpolitika, hardver/szoftver alrovatok is lehetnek. Minden egyes hír külön oldalon jelenik meg, és a főoldalról csak akkor van link egy híroldalra, ha a hír friss és fontos (ez már maga nem elhanyagolható információ), az aloldalakról minden abban a témában íródott hírre van link. A híroldalak HTML kódjai pedig tipikusan egy adatbázisból generálódnak: van egy fejlécük (általában hirdetéssel és a portál logójával), egy láblécük (hasonló reklám és/vagy önreklám tartalommal), közte pedig bal és/vagy jobboldalt egyéb nem feltétlenül a hírhez kapcsolódó információk (a portál legfrissebb hírei, egyéb szolgáltatásai, stb.). Ez a struktúra – lévén sok adatbázisból generált dinamikus része – megtévesztheti egyrészt az adatbázis frissességét ellenőrző algoritmust, hiszen egyrészt az ellenőrzések között mindig változik az oldal (a friss hírekre való linkelés miatt), viszont tartalma tulajdonképpen nem módosul, valamint másrészt sok irreleváns linket visz (vihet) be a linkstruktúrába. Ezesetben a legtöbb link irreleváns, csak az általában közvetlenül a cikk alatt elhelyezkedő „ajánlott linkek" vehetők relevánsnak. Ez tehát azt jelenti, hogy a domain-összehúzás alkalmazása jó közelítés, de látnunk kell, hogy egy kis információvesztés mindig fellép. A 2. ábrán is jól látható, hogy egy kisméretű portálon kevés friss hír is hogy megzavarja a linkstruktúrát.

Hol is kell a domainek közötti határokat meghúzni? Erre több cikk kereste a választ. Legátfogóbb eredmény talán a ([11]) cikk adta, amely egy döntési fát határozott meg tapasztalati úton. A cikk több ismérvét sorolja fel az oldalak egybe vagy külön tartozásának, például azt, hogy az URL host vagy domainneve azonos-e, tartalmaz-e tilde (~) karaktert, a két dokumentum hány százalékban tartalmaz azonos linkeket, illetve szerepel-e azonos emailcím az oldalakon. A keresőben ezt az empirikus eredményt alkalmaztuk, néhol kiegészítve a magyar webre vonatkozó részekkel (lásd bővebben [16]).

5. A linkstruktúrát használó rangsoroló algoritmusok

Ebben a fejezetben egy gráf pontjainak rangsorát megállapító algoritmusokat írunk le, ahol egy pontból egy másikba mutató él a kezdőpont végpontba vetett „bizalmát", a végpont tartalmának elismerését jelenti. Ezen belül is két algoritmuscsalád létezik, az egyik családba tartozó algoritmusok a *gráf összes pontját súlyozzák*, majd a súlyok rendezésével állapítják meg a sorrendet. Ezek a *globális algoritmusok*, míg a másik családot *kérdésfüggő rangsorolások*nak nevezhetjük, ami azt jelenti, hogy a rangsoroló algoritmus minden kérdésnél lefut, és ekkor csak egy *részgráf* csúcsait pontozza. Mindkét családnak megvan a maga előnye, ezt a későbbiekben látni fogjuk.

5.1. Globális algoritmusok

5.1.1. Egyszerű *PageRank* algoritmus

Tekintsünk egy n pontú gráfot, aminek pontjait rangsorolni akarjuk „fontosság" szerint, azaz minél több él mutat rá minél fontosabb pontokból, annál fontosabb lesz az adott pontunk. Ennek a gráfnak egy j pontjába mutató élek kezdőpontjainak halmazát jelöljük $B(j)$-vel (backlink): $B(j) = \{k : k \to j\}$! A *PageRank* algoritmus ([8]) a gráf összes pontjához rendel súlyt a következő képlet szerint:

$$\overline{PR}(i) = \sum_{j \in B(j)} \frac{\overline{PR}(j)}{N(j)},$$

ahol $N(j)$ a j. oldalon levő linkek száma, azaz az oldal ki-foka.

A *PageRank* súlyozás rekurzív definíciója:

$$\forall i \left[\overline{PR}(i)\right]_1 := \frac{1}{n}$$

$$\forall i \left[\overline{PR}(i)\right]_k := \sum_{j \in B(j)} \frac{\left[\overline{PR}(j)\right]_{k-1}}{N(j)}.$$

Ez más szavakkal azt jelenti, hogy $\overline{PR} = M^T \overline{PR}$, ahol az $M \in \mathbb{R}^{n \times n}$ mátrix m_{ij} eleme $\frac{1}{N(i)}$, ha van $i \to j$ link, különben 0. Erősen összefüggő gráf esetében az iteráció határértéke az M mátrix 1 *sajátérték*hez tartozó egyetlen *sajátvektor*a, ugyanis ekkor az M mátrix egy *Markov lánc átmenetvalószínűség-mátrix*a, amelynek stacionáris eloszlása épp kielégíti a fenti egyenletet.

Másképp kifejezve az így kapott súlyozás egy olyan **véletlen séta** stacionáris eloszlását adja, amelyet egyenletes eloszlás szerint indítunk, és minden lépésnél egyenletes eloszlás szerint választjuk az éppen aktuális oldalról kifelé mutató élek közül azt, amelyik mentén továbbhaladunk. Ez a séta egy egyenletesen véletlenül böngésző felhasználó viselkedését modellezi.

5.1.2. A módosított *PageRank* algoritmus

A gráfnak azt a részét, amely erősen összefüggő, de nem vezet ki belőle link, csak belépő élekkel van hozzákötve a gráf többi részéhez, lyuknak nevezzük. Ha van(nak) ilyen(ek) a gráfban, akkor az eze(ke)n kívüli pontok halmazának *PageRank*-je 0 lesz. Ez látható lesz a 7. fejezet egy példáján is, de egyszerűen végiggondolható, hogy egy véletlen bolyongás egy lyukba pozitív valószínűséggel lép be, viszont 0 valószínűséggel lép ki. Ez határértékben épp a kívül eső pontok 0 *PageRank* értékét jelenti. A valóságban nagyon sok ilyen dokumentumhalmaz van a weben.

Az ezt az anomáliát kiszűrő módosított képlet

$$\overline{PR}(i) = d \sum_{j \in B(j)} \frac{\overline{PR}(j)}{N(j)} + (1-d)\frac{1}{n},$$

azaz

$$\forall i \;\; \left[\overline{PR}(i)\right]_1 := \frac{1}{n}$$

$$\forall i \;\; \left[\overline{PR}(i)\right]_k := d \sum_{j \in B(j)} \frac{\left[\overline{PR}(j)\right]_{k-1}}{N(j)} + \frac{1-d}{n}.$$

Ebben a modellben a *PageRank* érték szintén egy *stacionáris eloszlás*, méghozzá egy olyan *véletlen sétáé*, amely a fentebbitől csak abban tér el, hogy minden lépésnél a véletlen sétát (1-d) valószínűséggel megszakítjuk, és egy egyenletes eloszlás szerint kijelölt véletlen helyre ugrunk a gráfban. Az *átmenetvalószínűség-mátrix* ezesetben $dM + (1-d)U$ lesz, ahol M az előzőekben már használt mátrix, míg $U \in \mathbb{R}^{n \times n}$ minden eleme $\frac{1}{n}$.

5.2. Kérdésfüggő rangsorolások

A *PageRank* algoritmusokkal ellentétben az ezután leírt algoritmusok nem súlyozzák az összes pontot, csak egy részét, amely az adott kereséstől függ. Ez egyrészt azért is előnyös, mert a futtatási költségek összesítve kisebbek lehetnek, mint a *PageRank* algoritmusnál. Viszont a *PageRank* egyszer és mindenkorra (pontosabban a következő adatbázis-frissítésig) meghatározza az összes oldal sorrendjét, addig a következő algoritmusok minden kereséskor lefutnak, és másmás (kérdésfüggő) eredményt adnak.

A későbbiekben azonban az is bizonyításra kerül, hogy ***előzetes szűrés*** nélkül algoritmusaink nem adnak értékelhető eredményt, azaz az előzetes szűrést azért kell alkalmazni, hogy a kiindulási halmazunk úgy keletkezzen, hogy abban a ***keresés témája*** már nagyon *jól reprezentálva legyen.* Az alább leírt algoritmusok ugyanis hajlamosak az alaphalmazban jól reprezentált témájú oldalakat felpontozni, és ha ez más, mint a keresés témája, akkor ez a hatás az algoritmusok rossz működését eredményezi. Az alaphalmaz meghatározását tipikusan a következő módszerrel végzik: ([9])

Először veszünk egy gyökérhalmazt (tipikusan egy szöveges kereső által kiadott első néhány találatot). Ezekhez hozzávesszük az ezekre mutató oldalakat, és azokat az oldalakat, amelyekre linkelnek. Így megkaptuk az összes dokumentumot, amely az alaphalmazban szerepel, ezekhez az oldalakhoz hozzávéve az összes élet, ami az eredeti gráfban is benne volt, azt a részgráfot kapjuk, amelyen a továbbiakban dolgozni fogunk, és amelynek a keresés témáját jól reprezentálnia kell. Tegyük fel, hogy az így kapott részgráf N csúcsú! Legyen továbbá az A ennek adjacencia mátrixa, amelyben $a_{ij} := 1$, ha van $i \to j$ link, különben $a_{ij} := 0$!

Ezen az alaphalmazon szeretnénk megkeresni a legjobb tartalmas oldalakat, amelyekre a legtöbb értékelhető link mutat (*authorities*), illetve a legjobb linkgyűjteményeket (*hubs*), amelyek a legtöbb tartalmas, és a legkevesebb rossz oldalra mutatnak.

Az algoritmusok kétfajta pontozást alkalmaznak: ***authority*** (a sok jó *rámutató* élből következő *tartalmasság* mértéke) és ***hub*** (a sok jó *előremutató* élből következő jó *linkgyűjtemény* tulajdonság mértéke) pontokat. Jó authority pontokkal a tartalmas oldalak fognak rendelkezni, amelyekre sok jó linkgyűjtemény mutat, míg magas hub pontokat azok a linkgyűjtemények kaphatnak, amelyek sok tartalmas oldalra mutatnak (és kevés másikra). Ezeket a fogalmakat az alábbi algoritmusok sorra finomítani fogják.

5.2.1. Kleinberg algoritmusa (*HITS*)

Kleinberg algoritmusa ([15], [9], *HITS*, Hypertext-Induced Topic Search) az előzőekben leírt alaphalmazt használja. Az ennek megfelelő részgráf adjacenciamátrixát A-val jelölve azt szeretnénk elérni, hogy az $\underline{a}$ authority- és a $\underline{h}$ hub pontszámokra igaz legyen, hogy $B(i) = \{k : k \to i\}$ és $F(i) = \{k : i \to k\}$ jelölések mellett

$$\forall i \ \underline{a}(i) = \sum_{j \in B(i)} \underline{h}(j),$$

$$\forall i \ \underline{h}(i) = \sum_{j \in F(i)} \underline{a}(j),$$

azaz

$$\underline{a} = A^T \underline{h},$$

$$\underline{h} = A \underline{a},$$

azaz

$$\underline{a} = A^T A \underline{a},$$

$$\underline{h} = A A^T \underline{h}.$$

Ez a modell magában hordozza azt, hogy egy dokumentum csak akkor lehet jó hub, ha sok jó authority oldalra mutat, és egy dokumentum csak akkor jó authority, ha sok jó hub linkel rá.

A pontszámok iteratív kiszámítása definiálható: a két súlyvektor legyen kezdetben

$$[\underline{a}]_0 := [\underline{h}]_0 := (1, \ldots, 1)^T = \underline{1},$$

majd az iteráció lépése:

$$\forall i \; [\underline{a}(i)]_k = \sum_{j \in B(i)} [\underline{h}(j)]_{k-1},$$

$$\forall i \; [\underline{h}(i)]_k = \sum_{j \in F(i)} [\underline{a}(j)]_{k-1},$$

azaz

$$[\underline{a}]_k = A^T [\underline{h}]_{k-1},$$

$$[\underline{h}]_k = A [\underline{a}]_{k-1}.$$

Az így kapott vektorokat minden lépés után normálni kell, mivel semmilyen norma esetén nem normatartó az iteráció, nem is ez a célja, hanem a relatív fontosságot méri csak. Kleinberg bizonyította, hogy a két vektor az $A^T A$ ill. az AA^T mátrixok legnagyobb sajátértékeihez tartozó sajátvektoraihoz tartanak (összefüggő gráf esetén).

Az algoritmus jelentését az alábbiakban foglalhatjuk össze: az iteráció első lépésében minden csúcs authority pontszámként megkapja a belé mutató élek kezdőpontjainak hub pontszámait (kezdetben ez mind 1-gyel egyenlők), majd hub pontszámként a belőle induló linkek végpontjainak authority pontszámait (szintén az összes kezdetben 1). Definiáljunk most két **irányítatlan gráf**ot, a G_a ill. G_h (authority-, ill. hub-) gráfokat, amelyek csúcsai az eredeti gráf csúcsai (az oldalak), az i és j pont között pedig annyi él van, ahány olyan csúcs (hub) van, amiből i-be és j-be is mutat link, ill. hány olyan csúcs (authority) van, amibe i-ből és j-ből is el lehet jutni. Egy (dupla) lépés során a authority pontszámként a G_a gráfban levő szomszéd csúcsok authority pontszámait kapja meg minden csúcs, és minden lépés után normálunk.

Megjegyzendő, hogy ezeknek a gráfoknak az adjacencia-mátrixa éppen az AA^T, ill. az $A^T A$ mátrix, illetve, hogy ez nem egy véletlen séta az authority- ill. hubgráfon, hiszen minden pont minden élen a teljes pontszámát küldi el, nem osztódik el a pontszám.

Két ekvivalens megvalósítását használtuk az algoritmusnak, ezek mindössze a kezdő pontszámokban és az alkalmazott normában különböztek. Az elsőben csupa 1-es indítást és maximum normát, a másodikban csupa $\frac{1}{n}$-es indítást és összeg normát használtunk.

5.2.2. A *SALSA* algoritmus

Az algoritmus ([18], Stochastic Approach for the Link-Structure Analysis) a *HITS*nél megismert alaphalmazt alkalmazza. A hasonlóság a két algoritmus között szembetűnő, a *SALSA* ugyanazon az authority- ill. hubgráfon operál, méghozzá uniform véletlen sétát valósít meg rajtuk.

Az algoritmusnak megfelelő véletlen sétában a web gráf élei mentén először egyenletes eloszlással hátralépünk, utána előre. Ez az authority-gráfban egy irányítatlan élen való egyenletes eloszlású lépésnek felel meg. Így egy M_a *Markov lánc*ot definiálunk az authority-gráf pontjain, azaz az összes csúcson. Az M_h *Markov lánc*nál pedig előrelépéssel kezdünk, utána hátra lépünk, azaz a hub-gráf élein lépkedünk.

Az M_a és M_h *Markov lánc*ok formális definíciójához a $B(i) = \{k : k \to i\}$ mellett szükségünk lesz a $F(i) = \{k : i \to k\}$ jelölésre. Az előző bekezdés szerint a megfelelő átmenetvalószínűségek a következők:

$$P_a(i,j) = \sum_{k:k\in B(i)\cap B(j)} \frac{1}{|B(i)|}\frac{1}{|F(k)|} \text{ ill.}$$

$$P_h(i,j) = \sum_{k:k\in F(i)\cap F(j)} \frac{1}{|F(i)|}\frac{1}{|B(k)|}.$$

E láncok stacionáris eloszlásainak kiszámítására definiálható egy iteratív algoritmus:

$$[\underline{a}]_0 := [\underline{h}]_0 := (1,\ldots,1)^T = \underline{\mathbf{1}},$$

majd az iteráció lépése:

$$[\underline{a}(i)]_k := \sum_j P_a(j,i)\,[\underline{a}(j)]_{k-1}\,, \text{ ill.}$$

$$[\underline{h}(i)]_k := \sum_j P_h(j,i)\,[\underline{h}(j)]_{k-1}\,.$$

Feltéve egy pillanatra, hogy a *Markov lánc*aink irreducibilisek, azaz az authority- és hub-gráfok összefüggőek, az állítható, hogy az egyensúlyi eloszlásokban két pont authority ill. hub súlyának aránya megegyezik az eredeti gráfban vett be- ill. ki-fokszámainak arányával. Az állítás abból következik, hogy irreducibilis *Markov lánc*nak egyértelmű a stacionáris eloszlása ($\underline{\mathbf{a}}$), és a fenti súlyarányokat feltéve az állítás ellenőrizhető a következőképpen:

Az irreducibilitás miatt egyértelmű stacionáris eloszlás ki kell elégítse, hogy

$$\forall i\ \underline{\mathbf{a}}(i) = \sum_j P_a(j,i)\underline{\mathbf{a}}(j).$$

Most B-vel az élek halmazát jelölve és feltéve az előzőek szerint, hogy

$$\forall i\ \underline{\mathbf{a}}(i) = \frac{|B(i)|}{|B|},$$

így számolhatunk:

$$\begin{aligned}
\underline{\mathbf{a}}(i) = \sum_j \underline{\mathbf{a}}(j) P_a(j,i) &= \sum_j \underline{\mathbf{a}}(j) \sum_{k \in B(j) \cap B(i)} \frac{1}{|B(j)|} \frac{1}{|F(k)|} = \\
&= \sum_j \frac{|B(j)|}{|B|} \sum_{k \in B(j) \cap B(i)} \frac{1}{|B(j)|} \frac{1}{|F(k)|} = \\
&= \sum_j \frac{|B(j)|}{|B|} \frac{1}{|B(j)|} \sum_{k \in B(j) \cap B(i)} \frac{1}{|F(k)|} = \\
&= \frac{1}{|B|} \sum_j \sum_{k \in B(j) \cap B(i)} \frac{1}{|F(k)|} = \\
&= \frac{1}{|B|} \sum_{k \in B(i)} \sum_{j \in F(k)} \frac{1}{|F(k)|} = \\
&= \frac{1}{|B|} \sum_{k \in B(i)} 1 = \frac{|B(i)|}{|B|}
\end{aligned}$$

Ennek megfelelően a leírt *iteratív algoritmus* lefuttatására tulajdonképpen *nincs szükség*, hiszen a *stacionáris eloszlás* az előbbi elméleti eredmény felhasználásával *közvetlenül számítható*. Ezzel együtt természetesen az is igaz, hogy az algoritmus *könnyen becsapható*, hiszen – az előző fejezetben leírtak szerint – az oldalunkra mutató linkek száma tetszőlegesen növelhető.

Itt jegyezzük meg, hogy a *SALSA* az egyenletes eloszlással indított ***HITS* algoritmus első lépésé**nek felel meg, ezután az első lépés után a *HITS* algoritmusban a *SALSA* stacionáris eloszlásának súlyai jelennek meg.

Az egyensúlyi eloszlás *iteráció nélküli közvetlen kiszámítását* (a fokszámokból) *pSALSA* algoritmusnak nevezik. Ez egy nagyon gyors és egyszerű algoritmus, mindazonáltal abból a feltételezésből táplálkozik, hogy majdnem minden keresésre összefüggő hub- és authority-gráfú alaphalmazokat generálunk. A *pSALSA* algoritmus azonban kiterjeszthető *több komponensű gráf*ra is.

Több komponensből álló gráf esetén az algoritmus csak abban a komponensben dolgozik, ahonnan indult a séta. Mivel az indulás egyenletesen lett választva, ezért egy adott komponensből való indulás valószínűsége a *komponens méretével arányos*, azaz az alapgráfot G-vel, komponenseit G_k-val, az i csúcs komponensét j-vel jelölve

$$a_i = \frac{|G_j|}{|G|} \frac{|B(i)|}{\sum_{\alpha \in G_j} |B(\alpha)|},$$

ahol a nevezőben levő összeg a *komponens összes éleinek száma.*

A *SALSA* algoritmus súlyozott élekkel is működik, és ugyanezeket az eredményeket adja, azzal a kitétellel, hogy a fokszámokat és az élszámokat az élek súlyait figyelembe véve számoljuk. Ez a hozzáállás ígéretes lehet ha a *SALSA* olyan javításait keressük, amelyek kiszűrik a sok, a kereső becsapására generált élek hatását.

5.3. A *HITS* és *SALSA* algoritmusok módosításai

5.3.1. A *HITS* algoritmus és a véletlen séták

Az előző algoritmusaink közül a *PageRank* és a *SALSA* véletlen sétákon, *Markov lánc*okon alapult, viszont a *HITS* algoritmus nem. Feltehetnénk a kérdést, hogy a *HITS* algoritmus is kapcsolatba hozható-e *Markov lánc*ok stacionáris eloszlásaival? A válasz persze igenlő, hiszen minden sztochasztikus vektorhoz (így a *HITS* algoritmus eredményéhez) létezik olyan *Markov lánc*, amelynek stacionáris eloszlása éppen az a vektor. A kérdés már csak az, hogy létezik-e olyan ezzel a tulajdonsággal bíró *Markov lánc* is, amelynek átmenetvalószínűségei az eredeti gráfból származtathatók?

A válasz – kissé meglepő módon – az, hogy a *HITS* algoritmus összes közbülső eredménye előáll az eredeti gráfból származtatható *Markov lánc* stacionáris eloszlásaként, bár az, hogy az első fél lépés után már igaz ez (ott a *SALSA* súlyok jelennek meg), már előrevetíti az eredményt. Vezessük be a következő jelöléseket: egy B illetve egy F lépésnek egy a webgráfban levő link követését nevezzük hátra illetve előre irányban. Ezek kombinációit is definiáljuk, például $BFBF = (BF)^2$ egy négylépéses sétát jelent a webgráfban. Az i pontból a j pontba vezető $(BF)^n$ séták halmazát jelölje $(BF)^n(i,j)$, az i pontból induló $(BF)^n$ séták halmazát jelölje $(BF)^n(i)$, továbbá az összes $(BF)^n$ séták halmazára használjuk magát a $(BF)^n$ jelölést! Az $(FB)^n$ séták halmazai hasonlóan értendők.

Most definiáljuk a következő két *Markov lánc*ot: az állapotok halmaza az összes csúcs, amely magában az alapgráfban is benne volt, míg két csúcs között pontosan akkor van él, ha az alapgráfban van köztük legalább egy $(BF)^n$ illetve $(FB)^n$ séta. Az átmenetvalószínűségek pedig legyenek:

$$\begin{aligned} P_a(i,j) &:= \frac{|(BF)^n(i,j)|}{|(BF)^n(i)|}, \quad \text{illetve} \\ P_h(i,j) &:= \frac{|(FB)^n(i,j)|}{|(FB)^n(i)|}. \end{aligned}$$

A definíciókból az látható, hogy

$$\begin{aligned} |(BF)^n(i,j)| &= (A^TA)^n(i,j) \quad \text{és} \\ |(FB)^n(i,j)| &= (AA^T)^n(i,j), \quad \text{és ezekből} \end{aligned}$$

$$\begin{aligned} |(BF)^n(i)| &= \textstyle\sum_j (A^TA)^n(i,j) \quad \text{és} \\ |(FB)^n(i)| &= \textstyle\sum_j (AA^T)^n(i,j). \end{aligned}$$

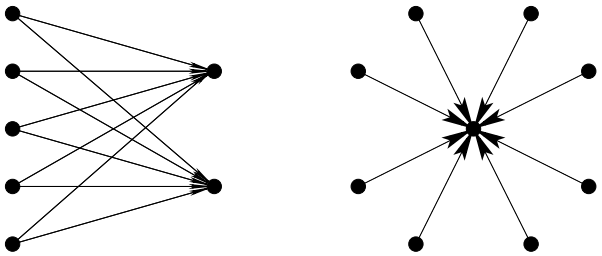
A *HITS* algoritmus n. iterációja után a pontszám vektorok normálás nélkül $(A^TA)^n\underline{\mathbf{1}}$ illetve $(AA^T)^n\underline{\mathbf{1}}$, azaz összeg normában ez ugyanaz, mint a megfelelő *Markov lánc*ok stacionáris eloszlása:

$$\underline{a}(i) = \frac{|(BF)^n(i)|}{|(BF)^n|}$$
$$\underline{h}(i) = \frac{|(FB)^n(i)|}{|(FB)^n|}$$

Elmondható tehát, hogy a *HITS* algoritmus végső pontszámarányai a csúcsokból induló hosszú BF illetve FB séták számainak arányától függ, aminek az a következménye, hogy nagyon erősen kötött alakzatok (teljes páros részgráfok) környékét az algoritmus kiemeli.

5.3.2. Páros részgráfok, közösségek

A *HITS* algoritmus legfőbb hibája, hogy nem feltétlenül a keresés témájához konvergál, ha az alaphalmazba egy **szorosabban kötődő téma** kerül (lásd 5.3.1. fejezet). Ennek kivédésére körültekintőbben kell előállítani az alaphalmazt, illetve jobban ki kell szűrni a nem informatív (pl. domainen belüli, navigációs, illetve reklám) linkeket.



3. ábra. Jó authoriy, illetve univerzálisan népszerű oldalak

Egy adott témát legjobban megjelenítő gráfalakzat egy, a gráf méretéhez képest nagy **teljes páros részgráf** (3. ábra). Ugyanis amennyiben egy témában (például számítógépprocesszorok) több tartalmas oldal (a processzorgyártó cégek oldalai), és sok, a téma iránt érdeklődő honlaptulajdonos van (a processzorokat tesztelő cégek, érdeklődő informatikusok), akkor ők az oldalaikon szerepeltetik a legjobb oldalakat linkként. Így nagy méretű teljes páros részgráfok tudnak kialakulni, amelyeket a közös téma köt össze.

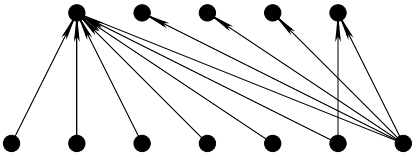
Kleinberget pont ez motiválta a *HITS* algoritmus kifejlesztésekor, algoritmusa a nagy teljes páros részgráfok kimutatását szolgálja. Egy, a témájában nagyon tartalmas oldalt is éppen az különböztet meg egy univerzálisan jó oldaltól, hogy sok teljes páros részgráfban szerepel.

Viszont ennek a stratégiának az a hátránya, hogy amennyiben az alaphalmazban több téma van reprezentálva, akkor *HITS* algoritmus ezek közül úgymond a legerősebbet választja ki, nem törődve a többivel (az iteráció csak a legnagyobb sajátértéket adja meg). Emiatt kell az alaphalmazunkat olyan gondosan megválasztani, hogy ne kerüljön bele sok, nem a témába vágó oldal, ez ugyanis a téma elcsúszását (topic drift) okozza.

Az algoritmus ugyanis az A^TA és az AA^T mátrixok legnagyobb sajátértékéhez tartozó sajátvektorát számítja ki. A tapasztalatok szerint ennek a vektornak a relatív nagy pontszámaihoz tartozó csúcsok egy témájú dokumentumokat határoznak meg. A mátrixok többi sajátértékei erre (és egymásra) mind merőlegesek, így egy, az előzőektől független témát jelentenek. A témák súlyait a sajátérték nagysága adja meg. Amennyiben azonban a legnagyobb sajátértékek nagyon közel vannak egymáshoz, a *HITS* algoritmus akkor is csak a legnagyobbat veszi figyelembe, a többi súly elveszik. Márpedig egy mátrix sajátértékei közel is eshetnek egymáshoz, mi több ez a hatás akár instabilitást is okozhat, hiszen a gráf (mátrix) kis változtatására a sajátértékek helyet is cserélhetnek (bővebben lásd a 8. fejezetben). Vannak próbálkozások az algoritmus olyan irányú módosítására, hogy az ne csak a legnagyobb sajátértékhez tartozó sajátvektort vegye figyelembe. Ezek a mátrixok szinguláris érték szerinti felbontását (Singular Value Decomposition, SVD, [1]) használják: $\forall A \in \mathbb{R}^{n\times m}\ \exists U \in \mathbb{R}^{n\times r(A)}$, $V \in \mathbb{R}^{m\times r(A)}$ mátrixok, melyek oszlopai ortonormáltak, valamint $\Sigma \in \mathbb{R}^{r(A)\times r(A)}$ diagonális mátrix, amelyekre $A = U\Sigma V^T$. Itt az $r(A)$ érték az A mátrix rangját jelenti. Ha ennek segítségével kiszámoljuk az A^TA mátrix értékét, akkor azt kapjuk, hogy $A^TA = V\Sigma^TU^TU\Sigma V^T = V\Sigma^2V^T$. Ez pedig a szimmetrikus mátrix Jordan-felbontása. Ha az A mátrixon elvégezzük az SVD felbontást, akkor a Σ mátrix módosításával, súlyozásával figyelembe vehetjük a többi sajátértéket is. Persze ez a kiszámítás jelentősen megnöveli a futási időt, látható ez a Jordan felbontással való közeli kapcsolatból is.

5.3.3. Az Átlagoló és Küszöb algoritmusok

A *HITS* algoritmusnál az a probléma is felmerül, hogy ha egy oldal nagyon sok rossz oldalra linkel, akkor nagyon sok hub pontszámot összegyűjthet, hiszen nem korlátozható a kimenő linkek száma. Ezt próbálja demonstrálni a következő egyszerű példa.



Képzeljük el azt a helyzetet, hogy $M+1$ linkgyűjtemény van, ezek $K+1$ másik (authority) oldalra linkelnek a következőképpen: az első M csak az elsőre linkel, míg az utolsó mindegyikre. Ezen kívül az utolsó előtti linkgyűjtemény az utolsó tartalmas oldalra is linkel. Ezesetben az első authority oldal a legjobb, ezt hozza ki a *HITS* algoritmus is, viszont a linkgyűjtemények közül az utolsónak kellene a legrosszabbnak lennie, mivel ez sok rossz oldalra is mutat, miközben a többiek csak jó oldalra linkelnek. A *HITS* algoritmus eredményeiben pedig ez pont a legjobb hub lesz (bővebben lásd a 7. fejezetet). Az effajta anomáliának a kijavítására használhatóak azok az algoritmusok, amelyek súlyozzák a kimenő linkeket, illetve ezekből tetszőlegesen sokat nem is vesznek figyelembe.

A fenti példában azért merült fel probléma, mert a linkgyűjtemények az összes belőlük kifelé mutató link végpontjának pontszámát megkapták. Azaz **megéri minél több linket összegyűjteni**, a nagyobb mennyiség az oldal

hub pontszámát javítja, azaz az algoritmus könnyen becsapható, ugyanis az így keletkezett oldal egyáltalán nem jó linkgyűjtemény, hiszen a felhasználó által áttekinthetetlen sok linket tartalmaz.

A Hub-Átlagoló algoritmus ezt úgy javítja ki, hogy a linkgyűjtemények pontszámának kiszámításánál azt veszi figyelembe, hogy *átlagosan milyen jó oldalakra mutat* az oldal. Ez tulajdonképpen azt jelenti, hogy az authority pontokat a *HITS* algoritmus, a hub pontokat pedig a *SALSA* algoritmus szerint adjuk. Végiggondolható, hogy egy oldalra mutató linkgyűjtemények számára nincs értelme megszorításokat tenni (hiszen erre az oldal szerzőjének nincs befolyása), viszont a mértékteken linkelést célszerű büntetni, hiszen az nulla költséggel elvégezhető, viszont a rengeteg rossz link a linkgyűjtemény minőségét inkább rontja. Ha ezt is figyelembe szeretnénk venni, akkor a Hub-Átlagoló algoritmus megbízhatóbb eredményt ad, mint egyrészt a tiszta *HITS*, másrészt a tiszta *SALSA* algoritmus (lásd 7. fejezet).

A másik lehetőség az ésszerűtlenül nagy hub oldalak felpontozásának elkerülésére, hogy csak az első néhány kifele mutató link mentén küldjük el az oldalunk hub pontszámát, hiszen az ezutáni linkeket nagy valószínűséggel már senki sem olvassa. Az a K szám, amely megadja, hogy hány linket veszünk így figyelembe, az algoritmus paramétere lesz, amelyet ezért K paraméterű Authority-Küszöb algoritmusnak nevezünk, mivel az authority pontszámok számításánál alkalmazza a csonkolást. Ennek az algoritmusnak kicsit nehézkes az alkalmazása, ugyanis a linkek sorrendjét is tárolni kell hozzá, hacsak nem ragaszkodunk egy adott K-hoz a kereső egész élete során, hiszen ekkor az e fölötti linkeket már a struktúra indexbe sem szükséges bevenni.

Itt kell még megemlíteni a Hub-Küszöb algoritmust, amely azt hivatott elkerülni, hogy jó oldalnak tartsunk egy olyan lapot, amelyre csak rengeteg rossz linkgyűjteményből hivatkoznak. Ez a fajta csalás szintén könnyen kivitelezhető, ellene úgy lehet védekezni, hogy a rá mutató élek közül csak azokat vesszük figyelembe, amelyek hub pontszáma az átlagos felett van.

Használjuk továbbá a Teljes Küszöb algoritmust, amely mindkét fenti küszöb algoritmust alkalmazza.

5.3.4. A BFS algoritmus

A *HITS* algoritmus pontszámainak az értéke az n. lépés után csak attól függ, hogy hány csúcshoz juthatok el n darab BF illetve FB lépésben (lásd 5.3.1. fejezet), a *SALSA* algoritmus viszont csak az első B lépést veszi figyelembe. Nyilván megint érdemes valamilyen középutat keresni, hogy a közelebb levő oldalak pontszámai nagyobb, a távolabbiaké kisebb súllyal számítsanak, mivel egy, a csúcsunktól távolabb levő nagyméretű teljes páros részgráf a csúcs súlyának exponenciális növekedését eredményezheti, ami nem kívánatos, mivel minél messzebb van egy ilyen alakzat, annál kisebb az esélye, hogy ugyanaz a témája, mint a csúcsunknak megfelelő oldalé. Tehát érdemes a *HITS* és a *SALSA* algoritmusok egyfajta hibridjét kipróbálni. Ez egy olyan algoritmuscsalád lesz, amelyben a különböző hosszú BF illetve FB sétákat különbözőképpen súlyozzuk. Egy ilyen megvalósítás az ún. BFS algoritmus ([7]), amelyben a következő

arányosság igaz:

$$a(i) \sim 2^{n-1}|BF(i)| + 2^{n-2}|(BF)^2(i)| + \cdots + |(BF)^n(i)|,$$
$$h(i) \sim 2^{n-1}|FB(i)| + 2^{n-2}|(FB)^2(i)| + \cdots + |(FB)^n(i)|.$$

Az algoritmus megvalósításában *HITS* lépéseket teszünk, majd az eredményt a megfelelő súllyal figyelembe vesszük, és továbblépünk a következő *HITS* lépésre.

Ez az algoritmus a *HITS* algoritmus megszorítása, viszont a *SALSA* algoritmus általánosítása, ugyanis míg a *SALSA* csak az első B lépéstől függ, addig a *HITS* n. iterációja pontosan az első n darab BF illetve FB lépéstől. Így a *HITS*hez minden BF lépést ki kell számítani, de csak a határértéküket értékeljük, itt minden csúcs közvetlen környezete nagyobb súlyt kap.

6. A web tartalom és gráf felépülésének egy modellje

Ebben a fejezetben egy olyan általános modellt mutatunk be, amely magába foglalja a **linkstruktúra** kialakulásán kívül a **dokumentumok kifejezéseinek**, a **dokumentumok témáinak**, a **felhasználók lekérdezéseinek** kialakulását is, és mindemellett meg tudja adni a lekérdezésekre az ***optimális választ*** egy valószínűségi modell segítségével [1].

Alapfeltevésként abból indulunk ki, hogy létezik véges, de kis számú, k darab *faktor*, amely az egész web belső szerkezetét jellemzi a következőképpen: minden a weben előforduló *téma* felírható egy k dimenziós vektorként, úgy, hogy két koordináta hányadosa azt adja meg, hogy az ezeknek megfelelő faktorok milyen arányban szerepelnek a témában. A weben található összes dokumentumhoz két k dimenziós sorvektort rendelünk hozzá, egy $A^{(p)}$-t, amely megadja, mely témában *authority*, és egy $H^{(p)}$-t, amely megadja, mely témában *hub* a p dokumentum. A vektorok iránya a témát jelenti, a vektorok nagysága pedig az authority illetve hub tulajdonság erősségét.

Feltesszük továbbá, hogy egy p oldalról egy q oldalra mutató **linkek száma** egy olyan X_{pq} *valószínűségi változó*, amelynek várható értéke az $A^{(q)}$ és $H^{(p)}$ vektorok *skaláris szorzat*a. Tehát minél közelebb van a két téma, és minél erősebb oldalakról van szó, annál nagyobb az esélye a linknek. Ez a bizonyos X_{pq} valószínűségi változó célszerűen $0-1$ értékű. Ha összesen n dokumentum van a weben, és az ezekhez tartozó sorvektorokból képezzük az A és H $n \times k$ méretű mátrixokat, valamint ezek szorzatát: $W := HA^T \in \mathbb{R}^{n \times n}$, akkor ennek a mátrixnak az (i,j) koordinátájú elemében az i-ből j-be mutató **linkek várható számá**t kapjuk. Az általunk megfigyelt webgráf ennek egy megvalósulása, egy ilyen *mátrixértékű valószínűségi változó*nak megfelelő eloszlásból vett egyelemű minta. Ennek megfelelően a mátrixát $\widehat{W}$-vel jelöljük.

Az összes lehetséges szóhoz is hasonló sorvektorokat rendelünk, $S_A^{(u)}$-t és $S_H^{(u)}$-t, amelyek koordinátái azt mutatják meg, hogy a szónak mennyi a várható előfordulás-száma egy olyan tiszta authority illetve hub oldalon, amelynek témája az aktuális koordinátához tartozó egységvektor. Ezek segítségével felírható egy u szónak egy p oldalon való *várható előfordulásainak száma*:

$$\langle S_A^{(u)}, A^{(p)} \rangle + \langle S_H^{(u)}, H^{(p)} \rangle$$

Az előző részhez hasonlóan definiálhatók az S_A és S_H $l \times k$ méretű mátrixok, amelyekkel a **szó-dokumentum mátrix** felírható:

$$S = HS_H^T + AS_A^T$$

A begyűjtött mintánkat analóg módon itt is $\widehat{S}$-sel jelöljük.

Végül a **lekérdezések**et is tudjuk hasonló módon modellezni: a keresést indító személy először egy témát jelöl ki magában, amelyet egy v sorvektorral jellemzünk, majd úgy választ a rendelkezésre álló szavak közül, ahogyan egy a

kijelölt témában tökéletes hub oldal választana, a $q = v^T S_H^T$ eloszlás szerint. Mi ebből sajnos megint csak az egyelemű mintát, $\widehat{q}$-t látjuk.

Kérdés, hogy hogy adjuk meg a **korrekt választ** a kérdésre. v ismeretében persze minden oldalhoz kiszámíthatjuk a keresés és az oldal távolságát az vA^T képlet szerint. Tehát eszerint rendezve kell az oldalakat visszaadni. Sajnos azonban a v sorvektort nem ismerjük, csak a $\widehat{q}$ kérdést.

Adott tehát a következő rendszerünk:

$$S = HS_H^T + AS_A^T, \tag{1}$$

$$W = HA^T, \tag{2}$$

$$q = v^T S_H^T, \tag{3}$$

kiszámítandó célfüggvényünk pedig

$$v^T A^T.$$

Ha a modellt egyszerűsítjük annyiban, hogy az (1) egyenlőség helyett a

$$S = HS_H^T \tag{4}$$

egyenlőséget tekintjük, akkor az állítható, hogy

$$v^T A^T = qS^{-1}W,$$

ahol a középső tényező a mátrix pszeudoinverzét jelöli. Ugyanis

$$qS^{-1}W = v^T S_H^T (HS_H^T)^{-1} HA^T = v^T S_H^T (S_H^T)^{-1} H^{-1} HA^T = v^T A^T.$$

Eszerint a képlet szerint a megfigyelt $\widehat{q}, \widehat{S}, \widehat{W}$ értékekkel számolhatunk.
Az általános esetben a következő algoritmus használható [1]:

- Számítsuk ki az SVD felbontását az $\widehat{M} = \left[\widehat{W}^T \| \widehat{S}\right]$ mátrixnak:

$$\widehat{M} = \left[\widehat{W}^T \| \widehat{S}\right] = U_{\widehat{M}} \Sigma_{\widehat{M}} V_{\widehat{M}}^T$$

- Keressük meg azt a legnagyobb m indexet, amelyre az m. és az $(m+1)$. szinguláris értékek különbsége még elég nagy!
- A többi szinguláris értéket elhagyva képezzük a $\widehat{M}$ m rangú SVD approximációját:

$$\widehat{M}_m = U_{\widehat{M}_m} \Sigma_{\widehat{M}_m} V_{\widehat{M}_m}^T$$

- Számítsuk ki az SVD felbontását a $\widehat{W}$ mátrixnak:

$$\widehat{W} = U_{\widehat{W}} \Sigma_{\widehat{W}} V_{\widehat{W}}^T$$

- Keressük meg azt a legnagyobb r indexet, amelyre az r. és az $(r+1)$. szinguláris értékek különbsége még elég nagy!
- A többi szinguláris értéket elhagyva képezzük a $\widehat{W}$ r rangú SVD approximációját:

$$\widehat{W}_r = U_{\widehat{W}_r} \Sigma_{\widehat{W}_r} V_{\widehat{W}_r}^T$$

- Minden adott $\widehat{q}$ kérdéshez számítsuk ki a

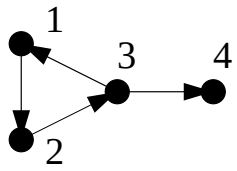
$$w = \left[0 \| \widehat{q}^T\right] \widehat{M}_m^{-1} \widehat{W}_r$$

vektort, ahol $\widehat{M}_m^{-1} = V_{\widehat{M}_m} \Sigma_{\widehat{M}_m}^{-1} U_{\widehat{M}_m}^T$ a $\widehat{M}_m$ mátrix pszeudoinverze.

A dokumentumokat a w vektor értékei szerint sorbarendezve kell rangsorolni. Az algoritmus helyességének bizonyítása, valamint az optimális paraméterbeállítások megtalálhatók a [1] cikkben.

7. Az algoritmusok működése néhány példán

Az algoritmusaink működését több gráfon teszteltük. Végső célként a web magyar részéről gyűjtött mintán szeretnénk kereséseket és rangsorolásokat végezni. Addig azonban az algoritmusok összes buktatóit fel kell térképeznünk, hogy a nagy gráfon való alkalmazáskor ne rossz eredménnyel szembesüljünk. Mivel az algoritmusok erényeit a bevezetésükkor ismertettük, ezért ebben a fejezetben a negatívumaikra koncentráltunk. Ezeket az anomáliákat kis példagráfok segítségével fogjuk demonstrálni. Ezek a kis példagráfok kiemelik azokat a részstruktúrákat a gráfban, amelyek a „hibát" okozzák, ezzel megpróbálják megnyitni azt az utat, amely az algoritmusok gyengeségeinek kijavítása felé mutatnak.



Első példánk a *PageRank* algoritmushoz kötődik. Az algoritmus egyik hiányossága, hogy sok nem erősen összefüggő gráfon nem működik jól. Ugyanis az algoritmusban minden lépésben az éleken szétküldi a csúcsokra írt pontszámokat. Egy olyan gráfban, amelyben van egy nulla kifokú csúcs (mint az ábrán) azonban az történik, hogy minden nem nulla kifokú csúcs pontszáma átadódik, viszont a nulla kifokúaké nem. Tehát minden lépésben elvesztünk pontokat, a kiosztott összpontszám nem lesz 1. A fenti gráfnál ez a hatás így érvényesül:

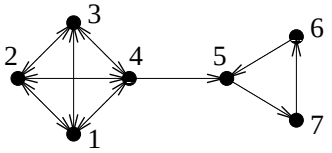
Sorszám	Kezdeti pontszámok	első iteráció után	második iteráció után	harmadik iteráció után	negyedik iteráció után
1	0.25	0.125	0.125	0.125	0.0625
2	0.25	0.25	0.125	0.125	0.125
3	0.25	0.25	0.25	0.125	0.125
4	0.25	0.125	0.125	0.125	0.0625
Összeg	1	0.75	0.625	0.5	0.375

Ennek a hatásnak a kivédésére általában kiszűrik a gráfból a nulla kifokú csúcsokat. Ez a módszer a következő példa szerint nem teljesen eredményes. Problémás továbbá az is, hogy egyes csúcsokat csak azért távolítunk el a gráfból, mert sehova sem linkelnek, ugyanis ezek a csúcsok ettől még nagyon tartalmasak lehetnek, nagyon sok más oldal linkelhet rájuk. Az elhagyás helyett például az alábbiakat lehet alkalmazni:

- felvenni egy hurokélt minden nulla kifokú csúcshoz
- a nulla kifokú csúcsok pontszámait minden lépés végén egyenletesen kiosztjuk.

A nulla kifokú csúcs egy csupa nullákból álló sort jelent a *PageRank* M mátrixában. Az első javaslat ezt egy egységvektorral helyettesíti, míg a második egy csupa $\frac{1}{n}$-es vektorral. A véletlen böngésző modellben ezek helyben maradást, illetve egyenletesen választott véletlen helyre való ugrást jelentenek. Ha helyben marad a böngészőnk, akkor az egyszerű *PageRank* algoritmusban az ilyen nulla kifokú csúcsok nagyon magukhoz fogják vonzani a pontszámokat, így inkább a második eljárás javasolt.

Második példánkban megmutatjuk, hogy bár a pontvesztés problémája a nulla kifokú csúcsok elhagyásával nem merül fel többet, viszont a pontok elszívását nemcsak egy ilyen csúcs okozhatja, hanem egy olyan részgráf, amelyből, mint halmazból nem mutat kifele él.



Az ábra jobb oldala mutat egy ilyen 3 csúcsú részgráfot, míg a bal oldalon egy maximális erősséggel kötő négy csúcsú rész látható. A *PageRank* algoritmus ennek ellenére a három csúcsú részgráf pontjait pontozza fel annak ellenére, hogy a bal oldali részgráf ennél jobban nem is köthet egymáshoz. A módosított *PageRank* algoritmussal kiszámoltunk néhány eredményt, az algoritmus d paraméterének függvényében:

d=1	első iteráció után	tizedik iteráció után	negyvenedik iteráció után	századik iteráció után
1	0.13095238804817	0.07131303846836	0.00927216932178	0.00015674947645
2	0.13095238804817	0.07131303846836	0.00927216932178	0.00015674947645
3	0.13095238804817	0.07131303846836	0.00927216932178	0.00015674947645
4	0.14285714924335	0.07633113861084	0.00992462877184	0.00016777953715
5	0.17857143282890	0.25479832291603	0.33331969380379	0.34485650062561
6	0.14285714924335	0.22427137196064	0.30831810832024	0.32066679000854
7	0.14285714924335	0.23066014051437	0.32062110304832	0.33383873105049

d=0.99	első iteráció után	tizedik iteráció után	negyvenedik iteráció után	századik iteráció után
1	0.13107143342495	0.07466895133257	0.02414103783667	0.01881407201290
2	0.13107143342495	0.07466895133257	0.02414103783667	0.01881407201290
3	0.13107143342495	0.07466895133257	0.02414103783667	0.01881407201290
4	0.14285714924335	0.07984372973442	0.02576031163335	0.02005849778652
5	0.17821429669857	0.24933451414108	0.31054437160492	0.31375992298126
6	0.14285714924335	0.22093778848648	0.29230934381485	0.30328533053398
7	0.14285714924335	0.22587707638741	0.29896289110184	0.30645403265953

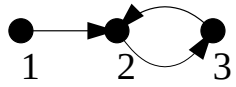
d=0.9	első iteráció után	tizedik iteráció után	negyvenedik iteráció után	századik iteráció után
1	0.13214285671711	0.09785078465939	0.08865854144096	0.08860762417316
2	0.13214285671711	0.09785078465939	0.08865854144096	0.08860762417316
3	0.13214285671711	0.09785078465939	0.08865854144096	0.08860762417316
4	0.14285714924335	0.10392615944147	0.09408707916737	0.09403257817030
5	0.17499999701977	0.21302194893360	0.22102472186089	0.22092877328396
6	0.14285714924335	0.19642576575279	0.21287536621094	0.21312102675438
7	0.14285714924335	0.19307377934456	0.20603720843792	0.20609498023987

d=0.5	első iteráció után	tizedik iteráció után	negyvenedik iteráció után	századik iteráció után
1	0.13690476119518	0.13300922513008	0.13300493359566	0.13300493359566
2	0.13690476119518	0.13300922513008	0.13300493359566	0.13300493359566
3	0.13690476119518	0.13300922513008	0.13300493359566	0.13300493359566
4	0.14285714924335	0.13793563842773	0.13793103396893	0.13793103396893
5	0.16071428358555	0.16256257891655	0.16256158053875	0.16256158053875
6	0.14285714924335	0.15269495546818	0.15270936489105	0.15270936489105
7	0.14285714924335	0.14777916669846	0.14778324961662	0.14778324961662

A táblázatból az szűrhető le, hogy a d paraméter csökkenésével az elszívó hatás is csökken, de megmarad. Ráadásul a $d = 0.5$ már teljességgel irrealisztikus eset, míg a realisztikus $d = 0.9$-nél még mindig nagyon erős az elszívó hatás.

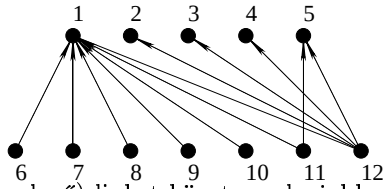
Célszerű tehát az ehhez hasonló gráfstruktúrákat azonosítani az algoritmus futtatása előtt, amely az összefüggő komponensek kiszámításával lehetséges. Ezt szerencsére polinom időben meg lehet tenni egy a mélységi keresésen alapuló algoritmus segítségével. A *PageRank* algoritmus csak erősen összefüggő gráfban fog jól működni, több komponens esetén mindig fennáll az elszívó hatás.

Következő példánkban a *PageRank* algoritmus egy instabilitási tulajdonságát vizsgáljuk, amely szintén a több összefüggő komponens esetén lép fel, és az algoritmus leállási kritériumának precízebb definiálására figyelmeztet.



Ha a *PageRank* algoritmust futtatjuk $d = 1$ paraméterrel a gráfra, akkor rögtön az első iteráció után 0 pontja lesz az első csúcsnak, a másodiknak 2/3, a harmadiknak 1/3. A továbbiakban az első csúcs végig pont nélkül marad, míg a másik két csúcson felváltva lesz 2/3 illetve 1/3 pont. Azaz az algoritmus végtelen ciklusba kerül. Ez a ciklizálás erősen összefüggő esetben nem történet meg, ennek ellenére érdemes arra figyelni, hogy mindig legyen egy maximális iterációszám, amely után az algoritmus leáll.

A következő példa a *HITS* algoritmus gyengéjére mutat rá, amely szerint egy nagyon sok rossz oldalra linkelő (tulajdonképpen ezért rossz) hub oldalt helyez a csak jó oldalra linkelő többi linkgyűjtemény elé.



Ebben a gráfban a 12-es számú linkgyűjtemény tekinthető a legrosszabbnak, hiszen nála a sok link közül csak egy értékelhető van, a többi rossz oldalakra mutat, és emiatt ezen az oldalon „elveszik" az egy jó link, míg a többi oldalon az egyetlen (és emiatt valószínűleg fő helyen levő) linket követve a legjobb oldalra jutunk. Erre a gráfra a *HITS* algoritmus a következő eredményt adja:

HITS

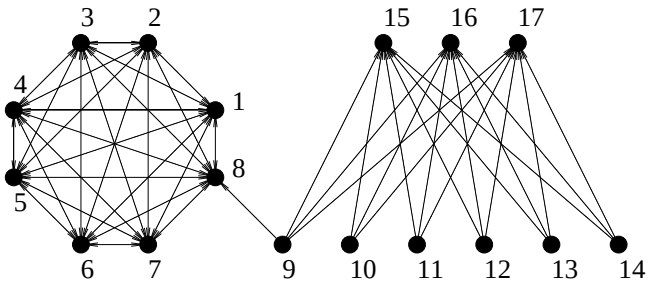
authority-pontszám	hub-pontszám	sorszám
0.45899677276611	0.00000000000000	1
0.11631865799427	0.00000000000000	2
0.11631865799427	0.00000000000000	3
0.11631865799427	0.00000000000000	4
0.19204725325108	0.00000000000000	5
0.00000000000000	0.11631868779659	6
0.00000000000000	0.11631868779659	7
0.00000000000000	0.11631868779659	8
0.00000000000000	0.11631868779659	9
0.00000000000000	0.11631868779659	10
0.00000000000000	0.16498717665672	11
0.00000000000000	0.25341939926147	12

Az ilyen, és ehhez hasonló példák miatt vezettük be a Hub-Átlagoló algoritmust, amelynek egyik lépése a *HITS* , míg másik lépése a *SALSA* algoritmus szerint adta a pontokat. Lássuk, ez és a *SALSA* algoritmus milyen eredményt ad!

	Hub-Átlagoló		*SALSA*	
authority-pontszám	hub-pontszám		authority-pontszám	hub-pontszám
0.58333331346512	0.00000000000000	1	0.58333331346512	0.00000000000000
0.08333333581686	0.00000000000000	2	0.08333334326744	0.00000000000000
0.08333333581686	0.00000000000000	3	0.08333334326744	0.00000000000000
0.08333333581686	0.00000000000000	4	0.08333334326744	0.00000000000000
0.16666667163372	0.00000000000000	5	0.16666668653488	0.00000000000000
0.00000000000000	0.16706444323063	6	0.00000000000000	0.08333332836628
0.00000000000000	0.16706444323063	7	0.00000000000000	0.08333332836628
0.00000000000000	0.16706444323063	8	0.00000000000000	0.08333332836628
0.00000000000000	0.16706444323063	9	0.00000000000000	0.08333332836628
0.00000000000000	0.16706444323063	10	0.00000000000000	0.08333332836628
0.00000000000000	0.10739856958389	11	0.00000000000000	0.16666667163372
0.00000000000000	0.05727924033999	12	0.00000000000000	0.41666668653488

Mint látható, a kívánt hatást a Hub-Átlagoló algoritmussal elértük, míg a gráf mind a *HITS* , mind a *SALSA* algoritmust megtéveszti. A Küszöb algoritmusok működését itt nem tudjuk bemutatni, hiszen ezekhez az itt használt struktúraindex nem elegendő, kellene tárolni azt is, hogy melyik link hányadik az oldalon.

Utolsóként egy már bemutatott gráfra (1. ábra) adjuk meg a *HITS* , a *SALSA* és a Hub-Átlagoló algoritmus pontszámait.



4. ábra. Szoros lokális közösség létrehozása a rangsorolás félrevezetése céljából

	HITS		*SALSA*	
authority-pontszám	hub-pontszám		authority-pontszám	hub-pontszám
0.12312405556440	0.12155684083700	1	0.09333326667547	0.09333326667547
0.12312405556440	0.12155684083700	2	0.09333326667547	0.09333326667547
0.12312405556440	0.12155684083700	3	0.09333326667547	0.09333326667547
0.12312405556440	0.12155684083700	4	0.09333326667547	0.09333326667547
0.12312405556440	0.12155684083700	5	0.09333326667547	0.09333326667547
0.12312405556440	0.12155684083700	6	0.09333326667547	0.09333326667547
0.12312405556440	0.12155684083700	7	0.09333326667547	0.09333326667547
0.12599344551563	0.12115348875523	8	0.10666661709547	0.09333326667547
0.00000000000000	0.01941726356745	9	0.00000000000000	0.05333342030644
0.00000000000000	0.00170627143234	10	0.00000000000000	0.04000009223819
0.00000000000000	0.00170627143234	11	0.00000000000000	0.04000009223819
0.00000000000000	0.00170627143234	12	0.00000000000000	0.04000009223819
0.00000000000000	0.00170627143234	13	0.00000000000000	0.04000009223819
0.00000000000000	0.00170627143234	14	0.00000000000000	0.04000009223819
0.00404605595395	0.00000000000000	15	0.08000017702579	0.00000000000000
0.00404605595395	0.00000000000000	16	0.08000017702579	0.00000000000000
0.00404605595395	0.00000000000000	17	0.08000017702579	0.00000000000000

Hub-Átlagoló

0.09333333373070	0.07639419287443	1
0.09333333373070	0.07639419287443	2
0.09333333373070	0.07639419287443	3
0.09333333373070	0.07639419287443	4
0.09333333373070	0.07639419287443	5
0.09333333373070	0.07639419287443	6
0.09333333373070	0.07639419287443	7
0.10666666179895	0.07486630976200	8
0.00000000000000	0.06951871514320	9
0.00000000000000	0.06417112052441	10
0.00000000000000	0.06417112052441	11
0.00000000000000	0.06417112052441	12
0.00000000000000	0.06417112052441	13
0.00000000000000	0.06417112052441	14
0.07999999821186	0.00000000000000	15
0.07999999821186	0.00000000000000	16
0.07999999821186	0.00000000000000	17

Itt az jegyezhető meg, hogy míg a *HITS* algoritmus „bedől" a gráfnak, a másik kettő már elfogadható eredményt ad.

Tanulságképpen az szűrhető le a fejezet végére, hogy az összes algoritmussal óvatosan kell bánni. A *PageRank* algoritmusnál az „elszívó" hatásra kell figyelni, a *HITS* algoritmusnál ugyanúgy. A *SALSA* algoritmus nagyobb ellenállást mutat ennek, de – mint ahogy azt korábban megmutattuk – nagyon egyszerűen befolyásolható, hiszen csak a fokszámokat számolja. A Hub-Átlagoló algoritmus az elvárásoknak megfelelően jól teljesít. Nagy reményt fűzünk továbbá a BFS algoritmushoz, amely még nem jutott tesztelésre kész fázisba.

8. Az algoritmusok stabilitása

Ebben a fejezetben azt a kérdést vizsgáljuk, hogy melyik algoritmus hogy viseli el, ha egy kis változás történik a linkgráfban. Az instabil algoritmusok erre nagy változással reagálhatnak. Nagy változás alatt itt azt kell érteni, hogy a sorrend elején (hiszen a sorrend végén már majdnem mindegy, milyen rendezés szerint jönnek az oldalak) egy oldal több helyet mozdul el, illetve a pontszámvektorok valamilyen normában nagyon eltérnek egymástól. Azt fogjuk megmutatni ([20]) alapján, hogy a *HITS* algoritmus bizonyos feltételek mellett instabil, míg ezzel összehasonlítva a *PageRank* algoritmus sokkal nagyobb stabilitást mutat.

Kezdjük egy példával, amely a *HITS* algoritmus által használt szimmetrikus $S = AA^T$ mátrixra feltételezi, hogy a két legnagyobb sajátértéke között δ különbség van. Azt fogjuk megmutatni, hogy kis δ esetén van a mátrixnak egy olyan (valamely mártixnormában mért) δ nagyságrendű perturbációja, amely a *HITS* végeredményében, azaz a legnagyobb sajátértékhez tartozó sajátvektorban nagy, valamely vektornormában mért $O(1)$ nagyságrendű változáshoz vezet.

Legyen az S szimmetrikus mátrix felbontása a sajátvektorokból álló U ortonormált mátrix és a sajátértékekből álló diagonális mátrix segítségével:

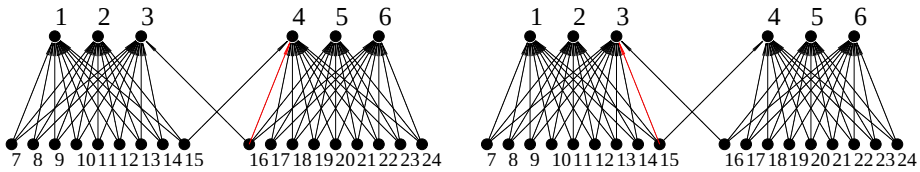
$$S = U \begin{pmatrix} \lambda & 0 & 0 \\ 0 & \lambda - \delta & 0 \\ 0 & 0 & \Sigma \end{pmatrix} U^T, \text{ és legyen}$$

$$\widehat{S} := S + 2\delta \underline{\mathbf{u_2}}\underline{\mathbf{u_2^T}} = U \begin{pmatrix} \lambda & 0 & 0 \\ 0 & \lambda + \delta & 0 \\ 0 & 0 & \Sigma \end{pmatrix} U^T \ !$$

Ekkor a mátrix megváltozásának Frobenius-nomája $\left\|\underline{\mathbf{u_2}}\right\|_2 = 1$ miatt 2δ, míg a két mátrix sajátvektorai ugyanazok, csak két legnagyobb sajátérték cserélődött meg, és ezáltal a *HITS* algoritmus határértéke megváltozott $\underline{\mathbf{u_1}}$-ről $\underline{\mathbf{u_2}}$-re, amely $O(1)$ nagyságrendű változás.

Ezt az instabilitást a gráfok nyelvére is le lehet fordítani, ez látható a 5. ábrán és az alatta levő táblázatban. Azt tapasztaljuk, hogy a linkstruktúra minimális megváltoztatása után a jobb és rosszabb authority-oldalak halmaza felcserélődik egymással, és ugyanez történik a linkgyűjtemények pontjainál. Egy él (az élek körülbelül 2%-a) megváltoztatásával egyes pontszámok körülbelül 60%-ot változnak.

Megállapíthatjuk, hogy a *HITS* algoritmus instabil abban az esetben, ha az AA^T mátrix két legnagyobb sajátértékének különbsége kicsi.



5. ábra. A gráf kis megváltozásának hatása a *HITS* algoritmusra

Authority pontszám	Hub pontszám		Authority pontszám	Hub pontszám
0.57140791416168	0.00000000000000	1	0.92455637454987	0.00000000000000
0.57140791416168	0.00000000000000	2	0.92455637454987	0.00000000000000
0.61803460121155	0.00000000000000	3	1.00000000000000	0.00000000000000
1.00000000000000	0.00000000000000	4	0.61803495883942	0.00000000000000
0.92455625534058	0.00000000000000	5	0.57140827178955	0.00000000000000
0.92455625534058	0.00000000000000	6	0.57140827178955	0.00000000000000
0.00000000000000	0.50786727666855	7	0.00000000000000	0.82174545526505
0.00000000000000	0.50786727666855	8	0.00000000000000	0.82174545526505
0.00000000000000	0.50786727666855	9	0.00000000000000	0.82174545526505
0.00000000000000	0.50786727666855	10	0.00000000000000	0.82174545526505
0.00000000000000	0.50786727666855	11	0.00000000000000	0.82174545526505
0.00000000000000	0.50786727666855	12	0.00000000000000	0.82174545526505
0.00000000000000	0.50786727666855	13	0.00000000000000	0.82174545526505
0.00000000000000	0.50786727666855	14	0.00000000000000	0.82174545526505
0.00000000000000	0.61803430318832	15	0.00000000000000	1.00000000000000
0.00000000000000	1.00000000000000	16	0.00000000000000	0.61803442239761
0.00000000000000	0.82174551486969	17	0.00000000000000	0.50786757469177
0.00000000000000	0.82174551486969	18	0.00000000000000	0.50786757469177
0.00000000000000	0.82174551486969	19	0.00000000000000	0.50786757469177
0.00000000000000	0.82174551486969	20	0.00000000000000	0.50786757469177
0.00000000000000	0.82174551486969	21	0.00000000000000	0.50786757469177
0.00000000000000	0.82174551486969	22	0.00000000000000	0.50786757469177
0.00000000000000	0.82174551486969	23	0.00000000000000	0.50786757469177
0.00000000000000	0.82174551486969	24	0.00000000000000	0.50786757469177

A *PageRank* algoritmussal szerencsére jobb a helyzet. Feltételezzük azt, hogy ha a véletlen böngésző 0 kifokú csúcsból az iteráció következő lépésében egy egyenletesen választott véletlen helyre kerül. Ekkor a módosított *PageRank* algoritmus egy az alapgráf pontjain értelmezett *Markov lánc* stacionáris eloszlását adja, melynek *átmenetvalószínűség-mátrix*a $dM + (1-d)U$, ahol d az algoritmus paramétere (lásd 5.1.2. fejezet), $U \in \mathbb{R}^{n\times n}$ az egyenletes áttérés mátrixa, míg $M \in \mathbb{R}^{n\times n}$ a gráf soronként normált adjacencia mátrixa (lásd 5.1.1. fejezet). Ha néhány csúcsot és a hozzá tartozó éleket bárhogyan megváltoztatunk a gráfban, akkor a következő korlátot tudjuk adni a *PageRank* vektor megváltozására:

Ha a megváltoztatott oldalak $i_1, i_2, \ldots, i_k$, és az új gráfhoz az $\widehat{M}$ mátrix tartozik, akkor a régi p és az új $\widehat{p}$ *PageRank* vektorokra igaz, hogy

$$\|p - \widehat{p}\|_1 \leq \frac{\sum_{j=1}^{k} p_{i_j}}{1-d}.$$

Bizonyítás: konstruálunk egy $(X_t, Y_t : t \in \mathbb{Z}^+)$ páros *Markov lánc*ot a dokumentumok alaphalmazán a következő szabályok szerint

- A két lánc ugyanonnan indul
- minden lépés kezdetén döntünk

– d valószínűséggel azt választjuk, hogy lépünk a gráfokon.
  * Ha a kiinduló időpontban a két lánc állapota ugyanaz volt, és ez az állapot nincs a megváltozott állapotok között, akkor egyenletesen lépünk a kimenő élek közül az egyik végpontjába, méghozzá mindkét lánccal ugyanoda.
  * Ha a kiinduló időpontban a két lánc állapota nem volt ugyanaz, vagy ugyanaz volt, de ez az állapot megváltozott, akkor a két láncon egyenletesen és függetlenül lépünk.
– $(1-d)$ valószínűséggel mindkét láncban megszakítjuk a lépéseket, és egy véletlen helyre ugrunk a gráfban, méghozzá mindkét láncban ugyanoda.

Ennek a két *Markov lánc*nak az *átmenetvalószínűség-mátrix*ai éppen a fent említett $dM+(1-d)U$ és $d\widehat{M}+(1-d)U$, tehát a stacionáris eloszlásaik p és $\widehat{p}$ lesznek. A két lánc állapotairól a következőket mondhatjuk:

$$\begin{aligned}
P(X_t \neq Y_t) &= P(X_t \neq Y_t|\text{megszakítás})P(\text{megszakítás})+ \\
&+P(X_t \neq Y_t|\text{nincs megszakítás})P(\text{nincs megszakítás}) = \\
&= 0 \times (1-d) + P(X_t \neq Y_t|\text{nincs megszakítás})d = \\
&= d[P(X_t \neq Y_t, X_{t-1} \neq Y_{t-1}|\text{nincs megszakítás})+ \\
&+P(X_t \neq Y_t, X_{t-1} = Y_{t-1}|\text{nincs megszakítás})] = \\
&= d[P(X_t \neq Y_t, X_{t-1} \neq Y_{t-1}|\text{nincs megszakítás})+ \\
&+P(X_t \neq Y_t, X_{t-1} = Y_{t-1}, X_{t-1} \in \{i_1, i_2, \dots i_k\}|\text{nincs megszakítás})] \leq \\
&\leq d[P(X_{t-1} \neq Y_{t-1}|\text{nincs megszakítás})+ \\
&+P(X_t \neq Y_t, X_{t-1} = Y_{t-1}, X_{t-1} \in \{i_1, i_2, \dots i_k\}|\text{nincs megszakítás})] \leq \\
&\leq d[P(X_{t-1} \neq Y_{t-1}) + P(X_{t-1} \in \{i_1, i_2, \dots i_k\}|\text{nincs megszakítás})] \leq \\
&\leq d[P(X_{t-1} \neq Y_{t-1}) + \sum_{j=1}^{k} p_{i_j}]
\end{aligned}$$

Tekintve, hogy $P(X_0 \neq Y_0) = 0$, azt állíthatjuk, hogy minden t-re

$$P(X_t \neq Y_t) \leq \frac{\sum_{j=1}^{k} p_{i_j}}{1-d},$$

amelyet a következő teljes indukciós lépéssel bizonyíthatunk:

$$P(X_{t+1} \neq Y_{t+1}) \leq d[P(X_t \neq Y_t) + \sum_{j=1}^{k} p_{i_j}] \leq d[\frac{\sum_{j=1}^{k} p_{i_j}]}{1-d} + \sum_{j=1}^{k} p_{i_j}] =$$

$$= \sum_{j=1}^{k} p_{i_j} \frac{d(2-d)}{1-d} \leq \frac{\sum_{j=1}^{k} p_{i_j}}{1-d}.$$

Tehát ez a becslés a sorozat határértékére is igaz lesz, ami azt jelenti ([2]), hogy a két stacionáris eloszlás variációs távolságára ($\frac{1}{2}\sum_{j=1}^{k} |p_i - \widehat{p}_i|$) is ez lesz a felső korlát. Ebből pedig az állítás egyből következik.

9. Összefoglalás

A dolgozatban azt tűztük ki célul, hogy a World Wide Web hálózatából kiszűrjük a legjobb oldalakat, amelyek egy néhány kulcsszavas keresés témájában a legrelevánsabbak. Ez a szűrés persze sok okból nagyon nehéz feladat, tekintve a feldolgozandó információ mennyiségét, dinamikusságát és a rengeteg csalási kísérletet.

A dolgozat először is áttekintést adott az Internet hálózat történetéről, struktúrájáról, benne az információcsere módjairól. Ezen belül különös figyelmet szentelt a *HTTP* protokollnak, illetve az e protokoll által közvetített dokumentumok leíró nyelvének, a *HTML*-nek és legfontosabb újdonságának, a linkelhelyezés lehetőségének.

A továbbiakban a dolgozat elemezte a World Wide Web felépülését, természetét, rámutatva fontos tulajdonságaira (nagy méret, változékonyság, heterogenitás), valamint az ezen való keresés kihívásaira (letöltés, indexelés, frissítés, rangsorolás). Szükség volt az információ-kinyerés (régi és új) formáinak megismerésére, hiszen a feladat adatbányászat jellegű, egy óriási információhalmazból egy nagyon kis méretű választ kell adni a feltett kérdésre. A szavak, szókapcsolatok elemzésén túl az új média lehetőséget nyújt az oldalak közti linkstruktúrában megjelenő kapcsolatok elemzésére, az oldalon meg nem jelenő metaadatok és a dokumentumokból hivatkozott multimédiás anyagok leíróinak felhasználására.

Ezek után át kellett tekinteni a weben előforduló tipikus megtévesztési módszereket, melyek segítségével a tartalomszolgáltatók a saját oldalaikat relevánsabbnak igyekeznek bemutatni, mint amilyenek valójában, azt remélve ettől, hogy több keresésben fognak az oldalaik az első találatok közt szerepelni, és így nagyobb forgalom koncentrálódik az oldalaikra. Ezeknek, a kereső számára irreleváns, sőt káros információknak a kiszűrése is fontos téma volt. Különös figyelmet kellett szentelni a lokálisan nagyon erős kötést létrehozó alakzatokra, valamint az egy egységbe tartozó oldalakra, amelyek nagyon sűrűn linkelnek egymásra. Ezeknek az egységeknek az azonosítása a rangsoroló algoritmusok alkalmazásának alapfeltétele, ha a csalásoktól valamennyire is védettnek szeretnénk érezni az eredményt.

Ezután a rangsoroló algoritmusok bemutatása következett. Megismerkedtünk a globális és a kérdésfüggő rangsorolások csoportjával. A globális rangsorolások között a *PageRank* algoritmust, és a módosított *PageRank* algoritmus került ismertetésre, amelyek pontozási rendszere egy véletlen böngésző viselkedésének modellezésén alapult, aki minden lépésben egyenletesen ugrik az épp aktuális oldalról egy linket követve egy másikra.

A kérdésfüggő rangsorolások két alaptípusa a *HITS* és a *SALSA* algoritmus. Ezekben már az oldal két tulajdonságát, a jó linkgyűjtemény (hub) és a tartalmas voltát (authority) is pontozzuk. A *HITS* algoritmus egy lépése az oldalak hub pontszámát a belőle induló linkek végpontjainak authority pontszámainak összegeként számítja ki, és viszont, míg a *SALSA* algoritmus ezeket a pontokat lenormálja a fokszámokkal, hogy egyfajta véletlen sétát kapjon. Bizonyítást nyert, hogy a *SALSA* algoritmus egyensúlyi eloszlása csak a csúcsok fokszáma-

itól függ, azaz könnyen félrevezethető, míg a *HITS* algoritmus a nagyon erősen kötő, de a témától idegen oldalakat hajlamos felértékelni. A két algoritmus rossz tulajdonságainak kiszűrésére bevezetésre kerültek az Átlagoló, a Küszöb, és a BFS algoritmusok.

A következő fejezetben bevezetett modell a web felépülését modellezi a dokumentumok szavainak beépülésétől kezdve a linkstruktúra felépüléséig, olyan feltételezéssel, hogy a weben előforduló összes téma felírható néhány faktor lineáris kombinációjaként. Itt minden kérdésre megadható a helyes válasz és egy algoritmus, mely a rendelkezésre álló adatokból ennek legjobb közelítését kiszámítja.

A továbbiakban néhány példa került bemutatásra, amely különféle negatív hatásokra irányította rá a figyelmet egyes rangsoroló algoritmusokkal kapcsolatban, majd tanáccsal szolgált ezeknek a hatásoknak megszüntetésére, csökkentésére.

A dolgozat zárásaként az algoritmusok stabilitásának elméleti vizsgátata történt meg. Az eredményekből kiemelhető a *PageRank* stabilitása, és a *HITS* algoritmus közel szimmetrikus alakzatoknál előforduló instabil jellege.

A jövőben a rangsoroló algoritmusokat valódi keresésekben szeretnénk alkalmazni, ennek előfeltétele az algoritmusok között egy súlyozás megtalálása, amely többé-kevésbé az összes negatív hatástól védi az eredményt, és minden algoritmustól a lehető legjobb tulajdonságokat örökli. Cél továbbá más algoritmusok tesztelése, ezek közül a dolgozatban leírt BFS algoritmushoz fűzzük a legnagyobb reményeket. Meg kell továbbá találni a globális és a kérdésfüggő rangsorolások közötti kapcsolódási pontot, hiszen érdemes mindkettőt alkalmazni annak ellenére, hogy nem ugyanazon az alaphalmazon dolgoznak. Izgalmas kérdés lehet továbbá az ismertetett web modellen alapuló kereső algoritmus gyakorlati kipróbálása. Összességében elmondható, hogy a cél ezután már a rangsoroló algoritmusok optimális kombinációjának meghatározása, amely stabil, nem túl nagy számításigényű, és a valódi adatokon is jó közelítő eredményt ad a keresésekre.

Hivatkozások

[1] Dimitris Achlioptas, Amos Fiat, Anna R. Karlin, and Frank McSherry. Web search via hub synthesis. In *IEEE Symposium on Foundations of Computer Science*, pages 500–509, 2001.

[2] D. Aldous. Random walks on finite groups and rapidly mixing markov chains. In *Seminaire de Probabilites XVII, 1981/82, Springer Lecture Notes in Mathematics 986, pp. 243–297*, 1981.

[3] Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, and Sriram Raghavan. Searching the web. *ACM Transactions on Internet Technology (TOIT)*, 1(1):2–43, August 2001.

[4] Albert-László Barabási and Réka Albert. Dynamics of complex system: Scaling laws for the period of boolean networks. *Physical Rewiew Letters*, 84:5660–5663, 2000.

[5] Albert-László Barabási, Réka Albert, and Hawoong Jeon. Mean-field theory for scale-free random network. *Physica A*, 272:173–187, 1999.

[6] Albert-László Barabási, Réka Albert, and Hawoong Jeong. Scale-free characteristics of random networks: the topology of the word-wide web. *Physica A*, 281:69–77, 2000.

[7] Alan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas. Finding authorities and hubs from link structures on the world wide web. In *Proceedings of the 10th World Wide Web Conference (WWW)*, pages 415–429, 2001.

[8] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[9] Soumen Chakrabarti, Byron E. Dom, S. Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, Andrew Tomkins, David Gibson, and Jon Kleinberg. Mining the Web's link structure. *Computer*, 32(8):60–67, 1999.

[10] Michal Cutler, Yungming Shih, and Weiyi Meng. Using the structure of html documents to improve retrieval. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California*, December 1997.

[11] Brian D. Davison. Recognizing nepotistic links on the web. In *AAAI-2000 Workshop on Artificial Intelligence for Web Search, Austin, TX*, pages 23–28. Artificial Intelligence for Web Search, Technical Report WS-00-01, July 30 2000.

[12] Windhager Eszter. *Webkeresők adatbázisainak frissítési stratégiái.* Diplomamunka, Eötvös Loránd Tudományegyetem Természettudományi Kar, Június 2002.

[13] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques.* The Morgan Kaufmann Series in Data Management Systems, Jim Gray, Series Editor. Morgan Kaufmann Publishers, August 2000.

[14] Lan Huang. A survey on web information retrieval technologies. Technical report, ECSL, 2000.

[15] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[16] Csalogány Károly. *Hipertext állományok rangsorolása a hiperlink gráf alapján.* Diplomamunka, Eötvös Loránd Tudományegyetem Természettudományi Kar, Június 2002.

[17] Kumar, Raghavan, Rajagopalan, Sivakumar, Tomkins, and Upfal. Stochastic models for the web graph. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 2000.

[18] R. Lempel and S. Moran. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. In *Proceedings of the 9th World Wide Web Conference (WWW)*, 2000.

[19] Uher Máté. *Webkeresők architektúrája. Robotok és web-gráf modellek.* Diplomamunka, Eötvös Loránd Tudományegyetem Természettudományi Kar, Június 2002.

[20] Andrew Y. Ng, Alice X. Zheng, and M. Jordan. Stable algorithms for link analysis. In *Proc. 24th Annual Intl. ACM SIGIR Conference*, 2001.