Egy vertikális keresőrendszer készítése

Orosz György

Pázmány Péter Katolikus Egyetem
Információs Technológiai Kar
e-mail: oroszgy@itk.ppke.hu

Kivonat A tanulmányban egy nyílt forrású alapokra épülő, nyelvi eszközökkel felvértezett vertikális keresőrendszer építésének lépéseit mutatom be. Az eljárás során sor kerül létező rendszerek összehasonlítására, egy kiválasztott kereső vizsgálatára, illetve a nyelvi modulok építésének bemutatására. A készített rendszert összevetem az eredetivel teljesítmény és relevancia szempontjából, majd megmutatom, hogy milyen területeken sikerült érdemben javítanom a működésen.

Kulcsszavak: információ-visszakeresés, keresőrendszerek, nyelvi kereső

1. Bevezetés

Keresőrendszerek használata mára már mindennapi rutinunk részét képezi. Internetes keresőkben egyre-másra tűnnek fel nyelvi eszközök, melyek célja, hogy minél pontosabb találatokkal szolgáljanak a felhasználóknak. Ezek az eszközök – nagy mennyiségű adatról lévén szó – többségében statisztikai alapúak.

Vertikális keresésről beszélünk akkor, ha a keresés az internet egy szeletén vagy valamilyen kritérium mentén szűkített korpuszon történik. Ez a kritérium leggyakrabban egy témakör, de lehet fájlformátum, dokumentumtípus stb. is. A továbbiakban bemutatom, hogy vizsgálataim alapján milyen lehetőségek kínálkoznak egy nyelvi vertikális kereső építésére.

Először áttekintem a nyílt forráskódú keresőrendszereket, megvizsgálom, milyen feltételek szükségesek egyáltalán egy ilyen projekt sikeréhez. Ezen elvárások mentén összehasonlítom a keresőket, majd egy választott képességeit megismerve a Magyar Országgyűlés plenáris üléseinek annotált korpuszához [5] készítek vertikális keresőt. Bemutatom a fejlesztés lépéseit, az integrált, illetve létrehozott új modulok képességeit, továbbá a készítés során megfogalmazott tapasztalatokat. Végül megmutatom, hogy az új rendszer hogyan teljesít: az egyes eszközökkel hol és milyen mértékben sikerült javítani az eredeti kereső működésén, különös tekintettel a relevancia szerinti eredményességen.

2. Keresőrendszerek összevetése

Munkám célja, hogy egy nyelvi eszközöket is használó általános célú vertikális keresőrendszert hozzak létre, majd vizsgáljam ennek működését. Ehhez alapul

egy létező nyílt forráskódú keresőt kerestem. A következőkben áttekintem, milyen alternatívák kínálkoztak.

2.1. Szempontok

Mielőtt elkezdtem volna a keresők közötti böngészést és azok tanulmányozását, megfogalmaztam azon szempontokat, melyek mentén a létező rendszereket vizsgálni kívánom:

- továbbfejlesztéshez használható API létezése, minősége
- projekt állapota: stabil verzió, aktív fejlesztők, dokumentáltság,
- integrált NLP eszözök száma, minősége,
- támogatott karakterkódolások (magyar nyelvű dokumentumok feldolgozásához szükséges legalább egy az alábbiakból: UTF-8, ISO 8859-2, WINDOWS CP-1250) volta.

2.2. Vizsgált keresőrendszerek

Munkám során igyekeztem a lehető legtöbb rendszert górcső alá venni. Ezek felkutatásához elsősorban az alábbi dokumentumokat vettem alapul:

- Emmanuel Eckard és Jean-Cédric Chappelierés - Free Software for research in Information Retrieval and Textual Clustering [4]
- Christian Middleton és Ricardo Baeza-Yates - A Comparison of Open Source Search Engines [2]

Így a következő rendszereket vizsgáltam: OpenFTS[1], Terrier[2], Lucene[3], Datapark Search Engine[4], Egothor[5], Xapian[6], ht://Dig[7] és Lemur/Indri[8]. Az Egothor, OpenFTS és a ht://Dig alkalmazásokat már munkám elején elvetettem, hiszen ezek fejlesztése évekkel ezelőtt abbamaradt, vagy fejlesztési dokumentáltságuk egyáltalán nem kielégítő.

A **Terrier** JAVA nyelven íródott, így UTF-8 támogatása biztosított, továbbá jól dokumentált API-val rendelkezik. Indexelés folyamán a rendszer egy pipelineon keresztül dolgozza fel az indexelendő kifejezéseket, mely jól használható (nyelvtechnológiai) modulok sorba kötéséhez. Fejlesztése folyamatos, a támogatás biztosított. Szótövező használatára – az említett csővezeték-kialakítás miatt – ad lehetőséget, de más nyelvi modulokat nem használ.

A **Lucene** egy teljes értékű keresőrendszer, mely szintén JAVA nyelven íródott, így a Terrierhez hasonlóan a magyar nyelv írásjeleit is tökéletesen kezeli. Jól

[1] http://openfts.sourceforge.net/
[2] http://terrier.org/
[3] http://lucene.apache.org/
[4] http://www.dataparksearch.org/
[5] http://www.egothor.org/
[6] http://xapian.org/
[7] http://www.htdig.org/
[8] http://www.lemurproject.org/

dokumentált alkalmazásprogramozási felülettel rendelkezik. Legfőbb erőssége, hogy jól skálázható és nagy teljesítményű indexelő program biztosítja az adatok gyors feldolgozását. Képes dátumok szerinti keresésre, mezők kezelésére, egyszerre több indexben való keresésre, továbbá többféle lekérdezéstípust is támogat. A projekt rendkívül jól támogatott, dokumentált. Nyelvi eszközök kapcsán az mondható el róla, hogy többféle szótövezőt és azok integrálását is támogatja.

A **Datapark Search Engine** egy C nyelven íródott kereső, melynek API-ja csak kis mértékben enged beavatkozni a rendszer működésébe. Ezt is elsősorban olyan esetekben használják, amikor a szoftvert egy komplex programba vagy weboldalba építik be. Némi dokumentáció is fellelhető a világhálón, de ez korántsem teljes. Több karakterkészletet is támogat, viszont nyelvi eszközökkel csak nehézkesen gyarapítható. Szótövezés csak Aspell, illetve Ispell alkalmazásokon keresztül valósítható meg, illetve szinonimák és mozaikszavak felismeréséhez szótárfájlok használatával biztosít lehetőséget.

A **Xapian** egy olyan eszközkészlet, melyet C++ nyelven írtak és támogatja Unicode karakterek használatát. Egy ráépülő népszerű keresőrendszer az **Omega**, mely integrált nyelvi eszközzel rendelkezik, úgymint szótövező (magyar nyelvű is!) és keresés közbeni szinonimahasználat. Képes még több adatbázis egyidejű indexként való használatára. A projekt jól dokumentált és folyamatosan karbantartott. A kereső alkalmazásprogramozási felületét is úgy tervezték, készítették, hogy elsősorban a program beágyazását tegye lehetővé, nem pedig a bővítését.

A **Lemur** eszközkészlet a Carnegie Mellon University és a University of Massachusetts támogatásával készült. Céljuk egy olyan keretrendszer létrehozása volt, mely nyelvmodellezési és adatbányászati kutatásokhoz jól használható. A program C++ nyelven íródott, de rendelkezik API-val más nyelvekhez is. Az angol nyelvi szövegeken kívül kínait és arabot is támogat (nyelvi eszközökkel is), továbbá beépített angol szótövezőkkel és mozaikszó-felismerővel is rendelkezik. A rendszer a dokumentumok feldolgozását pipeline-szerűen végzi, melyhez jól használható API-t biztosít, továbbá képes több indexfájl egyidejű használatára is. Beépített nyelvi eszközei: angol nyelvű szótövező (2 db), arab nyelvű tövező, betűszó-felismerés és stopwordtámogatás. Az **Indri** egy olyan keresőrendszer, mely a Lemur eszközkészletére épül. A fentieken kívül fontos tulajdonsága, hogy támogatja az UTF-8 kódolású dokumentumokat is.

A fentebb részletezett szempontok alapján az Indri keresőrendszer tűnt a legjobb választásnak, így a továbbiakban az ezzel való munkámat és eredményeimet mutatom be.

3. Integráció

A kereső készítése során a célom az volt, hogy megvizsgáljam, hogy egy létező rendszer nyelvi eszközökkel való kiegészítésére milyen lehetőségek mutatkoznak, illetve milyen eredményességgel lehetséges ez a munka. A fejlesztéshez a Magyar Országgyűlés plenáris üléseinek annotált korpuszát vettem alapul. Ezt tanulmányozva készítettem el a meglévő nyelvi eszközök integrációját, illetve hoztam létre újakat. Abból a hipotézisből indultam ki, hogy egy jól illeszkedő

lemmatizáló[9] modul és egy kontrollált szinonimaszótár[1] csak javíthat a kereső eredményességén. Ezeken kívül a következő eszközöket készíttetem el és integráltam az Indribe dátum szerinti, mértékegységek szerinti kereshetőség és személyek/felszólalók keresése. Az alábbiakban áttekintem a fejlesztés során tapasztalt nehézségeket, az ezekre adott megoldásokat illetve a használt módszereket.

3.1. Létező modulok integrációja

Az alábbiakban bemutatom, milyen létező eszközöket és hogyan használtam a keresőrendszer fejlesztéséhez.

A kapott **lemmatizáló** modul beépítése, a korábban említett pipeline-szerű feldolgozásnak köszönhetően, különösebb nehézségek nélkül működött. Az egyetlen probléma a modul és a rendszer különböző karakterkódolásai közti konverzió volt, amit az `iconv` alkalmazás segítségével oldottam meg. A kereső az indexelés folyamatában a dokumentumot mint szavak halmazát kezeli, és egy szóhoz egyetlen tövet enged tárolni. Így – mivel lehetőségem sem volt egy-egy szó kontextusának a vizsgálatára – minden szóhoz annak leggyakoribb és egyben legvalószínűbb szótövét tároltam. (Ezzel nyilván rontva a rendszer teljesítményét a fedést illetően.) Az Indri képes arra, hogy az index létrehozásakor megjegyezze a készítéskor használt lemmatizálót, így lekérdezéskor a keresőkifejezésen képes ugyanazt futtatni. Ebből kifolyólag az indexelés/lekérdezés folyamataiba, a tövező integrálása céljából összesen egy helyen volt szükséges beavatkozni.

A **szinonimaszótár** integrálása során azt találtam, hogy ennek a funkciónak a rendszer általi automatikus használata – a generált nagy mennyiségű zajos adat miatt – jelentősen ronthat [6] a kereső eredményességén. Így úgy döntöttem, hogy oly módon teszem elérhetővé ezt az eszközt, hogy a felhasználó kontrollálhassa annak kimenetét. Ennek módja a következő volt: az Indri lekérdezőnyelvét bővítettem egy új operátorral (`~`), melynek használatakor az utána álló szó vagy kifejezés szinonimáival kiegészítheti a felhasználó a keresőkifejezést. A felhasználói interfész létrehozásakor a kereső lekérdezőnyelvére tudtam támaszkodni, hiszen az több operátort is biztosít rokon értelmű szavak összevonására a kifejezés kiértékelése során. Hasonlóan a tövezőhöz, a modul integrációja során az egyetlen technikai probléma a karakterkódolások közötti konverzió volt.

A **tiltólistás szavak használata** egy elterjedt módszer keresőrendszerek használata során a nem kívánt – jelentős információtartalommal nem bíró – szavak keresésbőli szűrésére. Az általam használt alkalmazás két úton engedi szűrni az ilyen szavakat: egyszerű felsorolással, illetve egy, a pipeline-ba illeszkedő modul írásával. Az előbbit választva a szűrni kívánt szavak listájának alapjául egy szabadon elérhető kis méretű adatbázist[10] használtam, melyet a korpuszból kinyert szóhalmazzal bővítettem. Itt a kiválasztás emberi erővel történt az előforduló szavak leggyakoribb 1%-án.

[9] A rendszer elkészítéséhez a MorphoLogic Kft. moduljait használtam.
[10] http://snowball.tartarus.org/algorithms/hungarian/stop.txt

3.2. Saját fejlesztésű modulok

A rendelkezésre álló korpusz vizsgálata során azt láttam, hogy a dokumentumokban számos olyan információtartalom áll rendelkezésre, melyek kereshetővé tétele – feltételezésem szerint – javít a keresések pontosságán. Ezen információk egy része az annotált korpuszban már jelölve volt, míg másokat a készített alkalmazás tett jelöltté. Így a keresőben most lehetőség nyílik a dokumentumokban fellelhető dátumok és mértékegységek *egységes* keresésére. Ezen kívül a korpusz struktúrájában rendelkezésre álltak a felszólalások adatai, így az egyes felszólalók neve is. Ezt az információt felhasználva lehetőség nyílik személyekre való keresésre is.

A korpuszban található **dátumok** használata során azt tapasztaltam, hogy számos helyen sokféle típusú és formájú dátumokat használ az író. Így célom az volt, hogy a készítendő modul működése testreszabható legyen, és a lehető legtöbbféle formában megjelenő egységeket legyen képes felismerni, kereshetővé tenni. A korpuszban található dátumokat az alábbi csoportokra osztottam: pontosság szerint létezik évszázad pontos, évtized szerint pontos, évre pontos, hónap vagy évszak pontos, napra pontos, órára, percre, napszakra stb.-re pontos. Környezetfüggőség szerint létezik környezetfüggő[11], illetve környezetfüggetlen[12].

Az Indri rendelkezik dátumok kezelésének képességével. Ez a funkció nem csak az index létrehozásakor elérhető, hanem lekérdezésekkor is kitűnően használható (pl.: időintervallum lekérdezések futtatása). Indexeléskor a rendszer csak olyan dátumokat képes tárolni az adatbázisban, amiket a felhasználó a feldolgozandó dokumentumban dátummezőként előre jelzett. (Ezek közül is csak a környezetfüggetlen, napra pontos, és angolszász helyesírásúakat.) Az általam adott megoldás két részből áll. Az első egy olyan előfeldolgozó alkalmazás, mely az annotált korpuszban újabb annotációkat vezet be, míg a második ezeket a jelöléseket feldolgozva rögzíti azokat az indexbe az eredeti dátumkezelő funkció felületét használva. Ahhoz, hogy minél több dátumformát kezeljen a program, ezek megadásait a felhasználó kezébe adtam. Így egy konfigurációs fájl használatával a rendszer az ott megadott formákból egy-egy felismerő reguláris kifejezést készít, melynek segítségével jelöli és értelmezi a dátumokat. A megoldás sajnos magában hordozza a reguláris kifejezések korlátait, így az alábbi jelenlévő korlátok továbbra is élnek:

- csak környezetfüggetlen és napra pontos dátumokat vagyunk képes feldolgozni,
- gondolnunk kell a dátumok ragozott alakjára, és a konfigurációs fájlban megadni az ezeknek megfelelő dátumformákat,
- nem használhatunk ütköző[13] dátumformákat.

[11] Környezetfüggőnek nevezek olyan dátumokat, melyek csak a szövegkörnyezetükben értelmezhetőek pl.: tegnap, előző héten.

[12] Környezetfüggetlennek nevezek minden olyan dátumot, mely önmagában is értelmezhető pl.: 1986.04.30.

[13] Ütközőnek nevezek két dátumformát, ha létezik olyan dátum, ami mindkét formának megfelel.

Mennyiségek használatának kialakításakor támaszkodhattam az Indri azon funkciójára, hogy rendelkezik ún. numerikus mezők kezelésének lehetőségével. A megvalósított kiegészítő modul a dátumfelismeréshez hasonlóan két részből áll. Az első rész azt hivatott szolgálni, hogy a forrásfájlokban újabb annotációkat hozzon létre, melyek mennyiségeket jelölnek. Az előzőekhez képest itt annyival nehezebb a feladat, hogy tudnunk kell különbséget tennünk különböző mértékek között. Ezt úgy sikerült elérnem, hogy az annotálás folyamán különböző mértékek más-más címkével jelölődnek.

A magyar nyelvben a jelzős szerkezeteknek kötött a formája: a jelző megelőzi a jelzett szót. Mivel így tekinthetünk egy mennyiségi kifejezésre is, ezért feltételeztem, hogy egy mértékegység feldolgozásakor az azt megelőző szóval vagy számmal összetartoznak. A felismerő modul ebből az alapötletből kiindulva működik, azonban akadnak olyan esetek, mikor ez a hipotézis nem alkalmazható. Egy ilyen gyakori eset, amikor a mértékegységről mint fogalomról ír a szerző. Pl.: ” . . . az euró bevezetése. . . ” Egy másik nehézség, amivel a program fejlesztése során találkoztam, az, hogy az országgyűlési naplókban gyakoriak az olyan pénzösszegeket kifejező mennyiségek, melyekben vegyesen használnak betűvel és számmal írt mennyiséget, pl.: „70 milliárd forint”. A rendszer végül úgy lett kialakítva, hogy intelligensen felismerje és kezelje az ilyen formákat is.

A második modul, mely beépül a szövegfeldolgozási láncba, a mezők feldolgozásáért, reprezentálásáért felelős. Ez túllépve az eredeti kereső korlátain képes valós értékeket is tárolni, továbbá fontos tulajdonsága, hogy egységes (SI) formában reprezentálja az azonos típusú mértékeket. A készített modul egy gyakorlatban is sokat alkalmazható funkcionalitása, hogy képes betűvel írt – magyar nyelven megfogalmazott – számok felismerésére, átváltására. A modul nem kezel olyan törtszámokat, melyek hagyományos – tehát nem tizedes – alakban fordulnak elő. Az intervallumként megadott értékeknél a program a várttól eltérően működik, mivel csak az intervallum egy végét teszi kereshetővé.

A felsorolt eszközök elkészítésén kívül a kereső által nyújtott funkciókkal élve lehetővé tettem még a dokumentumokban fellelhető (annotált) **nevek, személyek, felszólalások, felszólalók visszakeresését**. Ennek megvalósításához felhasználtam az Indri indexelőjének azon funkcióját, hogy képes mezőket tárolni az indexben. Rendelkezésemre állt a korpuszban nagy mennyiségű annotált névelem[14] (főleg személynevek), így ezek indexelése kézenfekvő volt. Ezek az elemek jellemző módon a felszólalásoknál vannak jelölve (a felszólaló személye), de ezeken kívül az egyes beszédekben hivatkozott személynevek is sok esetben elérhetőek.

4. Eredmények

Ebben a fejezetben célom, hogy megvizsgáljam a kiegészített keresőrendszert, és több szempont mentén összehasonlítsam az Indrivel. Nyilvánvalóan nem tudom az összes új modul hatását teljes körűen mérni, mert néhányuk olyan új funkci-

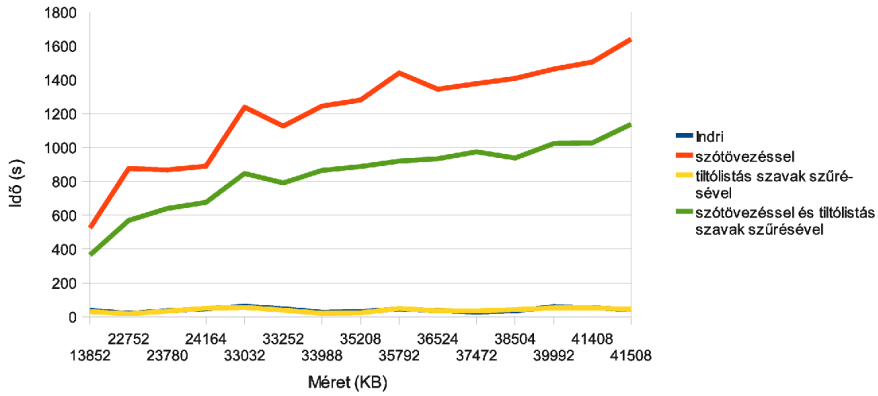
[14] Named Entity

onalitást ad az eszköznek, melyek nem vagy alig összehasonlíthatóak az eredeti program működésével.

4.1. Teljesítmény

Első lépésben azt vizsgáltam meg, hogy a keresőrendszer teljesítménye hogyan változik az általam készített eszközök hatására. Ez három helyen érhető tetten:

- indexelési idő változása,
- index méretének változása,
- keresési idő változása.

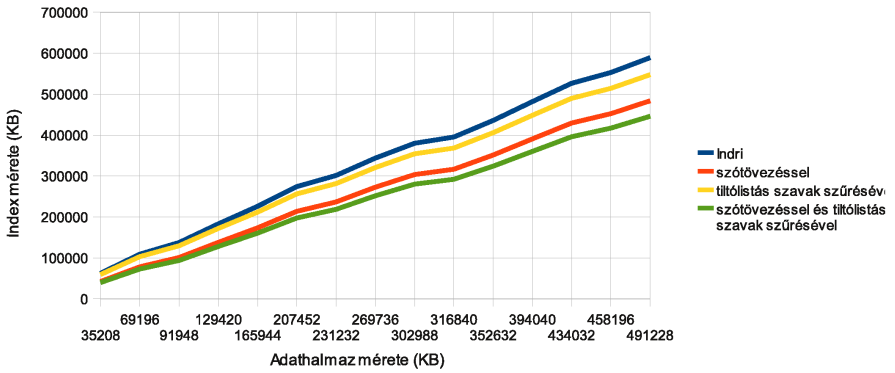


1. ábra. Indexelési idő az adathalmaz méretének függvényében

Indexelési idő. Az indexelési idő mérésénél a módszer az volt, hogy a dokumentumhalmazt valamilyen szempontok mentén csoportosítom, és mérem a rendszer indexeléssel töltött idejét az egyes halmazokon. Az ábrán jól látszik, hogy a szótövezés jelentősen lassította az indexelés folyamatát, viszont az is megállapítható, hogy a tiltólistás szavak szűrésével ez a teljesítménybeli visszaesés mérsékelhető.

Index mérete. Inkrementális indexelést végezve azt vizsgáltam, hogy hogyan változnak a készített indexek méretei, az eszközök használatának, illetve az adatmennyiség függvényében. Ennek a mérésnek az eredményét szemlélteti a 2. ábra.

Mint arra számítottam, az index mérete csökkent szótövezés és stopword-szűrés esetén. Tiltólistás szavak szűrésének alkalmazásakor, ez annak az egyszerű ténynek a következménye, hogy kevesebb kifejezés kerül az indexbe. Szótövezéskor pedig egy szó különböző ragozott alakjai ugyanahhoz a kifejezéshez tárolódnak. Az így nyert tárhelynövekedés a program eredeti működéséhez viszonyítva (átlagosan):



2. ábra. Index mérete az adathalmaz méretének függvényében

– szótövezéssel 22%,
– tiltólistás szavak szűrésével 6%,
– szótövezéssel és tiltólistás szavak szűrésével 28%.

Így tehát megállapítható, hogy e két eszköz használatával érzékelhetően lehet javítani egy keresőrendszer tárhelygazdálkodásán.

4.2. Relevancia

Módszer. A használt keresőrendszer eredménye egy rendezett lista, így az eredményesség mérésére az F-mérték csak korlátozott mértékben alkalmas. A kereső működéséhez jobban illeszkedő metrikák közül [1,3] az egyik legjobb tulajdonságokkal bíró az ún. *Mean Average Precision*[15]. Ezt a következő módon számolhatjuk: $\frac{\sum_{r=1}^{N}(P(r)\cdot \mathrm{rel}(r))}{R}$, ahol N a találati listában szereplő elemek száma, $rel(x)$ értéke 1, ha az x-edik elem a listában releváns, egyébként 0, $P(x)$ a pontosság a találati lista első x elemére szűkítve, R pedig a tárban található releváns dokumentumok számossága. A metrika szerint egy lekérdezés akkor 100%-os, ha az összes releváns elemet visszaadja, és ezek a találati lista legelején vannak (tehát nincs olyan dokumentum, ami nem releváns és megelőz egy relevánsnak mondottat). Látható, hogy ez a módszer nem érzékeny a zaj növekedésére, ha az a találati lista végén történik.

A mérésekhez minden esetben megfogalmaztam egy természetes nyelvű információigényt, majd azt az adott eszközökkel a lehető legpontosabban kódoltam. Mindehhez elkülönítettem a korpusz egy részhalmazát, mely elemeiről tudtam, hogy melyik milyen információigényhez hogyan kapcsolódik. A teszteket két indexen futtattam, összesen háromfélét[16]:

[15] A továbbiakban MAP.
[16] A továbbiakban 1., 2., illetve 3. mérés.

1. egyszerű[17] indexen, nyelvi eszközöket nélkülöző lekérdezéseket,
2. tövezett indexen a szótövezés és szinonimahasználat által nyújtott lehetőségeket használó lekérdezéseket,
3. tövezett indexen az előbbieken kívül az intelligens dátum- és mértékkeresés, illetve a személyek keresése használatával megfogalmazott lekérdezéseket.

A mérések készítése során felmerülő probléma, hogy hogyan is lehet objektíven leképezni az információigényt a kereső lekérdezőnyelvére. Érdekes kérdés, hogy mit feltételezek a felhasználóról:

- Mit tud a rendszerről? Mennyire ismeri a kereső lekérdezőnyelvét?
- Milyen forrásból fogalmazza meg a keresőkifejezést? (Az információigényt reprezentáló kérdés szavaiból vagy annak tágabb kontextusából?
- Milyen, a kérdéshez kapcsolódó többletinformációval rendelkezik a felhasználó?)
- Hány lekérdezést engedek futtatni? (Egy lekérdezés eredményének felhasználásával esetleg jobb keresőkifejezést lehet megfogalmazni.)

A tesztek futtatása során azzal a feltételezéssel éltem, hogy a felhasználó a lehető legteljesebben ismeri a lekérdezőnyelvet, illetve adott esetben rendelkezik a kérdéshez kapcsolódó plusz információval, és ezeket akár tágabb kontextusban is képes megfogalmazni. Viszont megszorításként csak egy lekérdezést futtattam egy-egy információigény megválaszolására.

Mérési eredmények. Korábbi méréseim [6] során már megmutattam, hogy szótövező használata érzékelhetően tudja növelni a kereső relevancia szerinti eredményességét, persze egyes esetekben (pontatlan keresőkifejezések használatakor) a zaj is növekszik. Méréseim megerősítettek abban a feltevésben, miszerint a lemmatizálással kombinált megfelelően kontrollált szinonimahasználat jelentős mértékű javulást tud eredményezni. A tesztek során az 1. mérésben az Indri eredményessége a MAP metrikával 71%-os volt. Ez azt jelenti, hogy az esetek nagy részében a releváns elemek a találati lista elején foglaltak helyet. Ezen sikerült javítani a 2. mérés folyamán, ami ezzel a metrikával 81%-os eredményt jelent. Szinonimákat használva azt tapasztaltam, hogy a keresőkifejezés nagyobb információtartalommal bíró szavaira alkalmazva jelentősen tud javítani az eredményeken (akár 20-50%-ot is). Ellenben egy-egy olyan szó esetén, mely a korpuszt tekintve gyakorinak mondható (pl.: beszéd, felszólalás), szinonimákat használni nem érdemes, mert ezekkel csak több zajt generálunk. A 3. esetről általánosan elmondható, hogy összességében itt is sikerült még további javulást eredményezni, így az említett módszerrel 85%-os eredményt kaptam. Mint várható volt, itt azon keresések eredményességén sikerült érdemben javítanom, amik valamilyen személlyel, mennyiséggel, időponttal kapcsolatos információt tartalmaztak. Elmondható még, hogy ha a felhasználó az információigényéhez kapcsolódóan többlettudással rendelkezik, úgymint 'kinek a felszólalásában található a kívánt információ?', 'kb. mikorra tehető a felszólalás időpontja?', 'milyen időpontok kap-

[17] szótövezést nem használó indexelővel készített

csolódnak a kérdéshez?', 'milyen mennyiségek (és milyen nagyságrendben) hozhatóak kapcsolatba a kérdéssel?', akkor további javulással számolhatunk. Összességében megállapíthatom, hogy a használt eszközökkel egy felhasználóbarátabb, relevánsabb találatokat produkáló rendszert sikerült építenem.

5. Összefoglalás

A cikkben bemutattam, hogy miként sikerült egy nyílt forráskódú keresőrendszert nyelvi eszközökkel bővíteni: várakozásaimnak megfelelően csak kisebb technikai akadályokba ütközött a rendszer bővítése. Az eredeti tervvel ellentétben nem volt lehetséges NLP-eszközök teljes integrációja kizárólag az Indri API-ját használva, így egyes esetekben módosítanom kellett a rendszer belső működését. Mindenképpen sikernek könyvelem el, hogy a rendszer új, eddig nem elérhető eszközökkel bővült, továbbá azt, hogy ezek javítanak, javíthatnak a kereső eredményességén. Az indexelési időben felmerült negatív eredményekről elmondható, hogy azok a végfelhasználót nem érintik, tehát érdemben nem rontottam a rendszer teljesítményén.

Hivatkozások

1. C. J Van Rijsbergen: Information Retrieval. London, Butterworths (1979).
2. Christian Middleton, Ricardo Baeza-Yates: A comparison of open source search engines. Jelentés, Universitat Pompeu, Fabra Department of Technologies (2007).
3. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze: Introduction to Information retrieval. New York, Cambridge University Press (2008).
4. Emmanuel Eckard, Jean Cédric Chappelier: Free Software for research in Information Retrival and Textual Clustering. Jelentés, Ecole Polytechnique Fédérale de Lausanne (2007).
5. Nagy István Zoltán: A Magyar Országgyűlés plenáris üléseinek annotált korpusza. Diplomamunka, Pázmány Péter Katolikus Egyetem, Információs Technológiai Kar, Budapest (2008).
6. Orosz György: Nyelvtechnológiai alkalmazások integrálása keresőmotor(ok)ba. Diplomamunka, Eötvös Loránd Tudományegyetem, Informatikai Kar, Budapest (2010).
7. Prószéky Gábor, Kis Balázs: Számítógéppel emberi nyelven. Bicske, SZAK Kiadó (1999).
8. Thorsten Brants: Natural language processing in information retrieval. Konferenciakiadvány, 14th Meeting of Computational Linguistics in the Netherlands (2003).
9. Trevor Strohman, Donald Metzler, Howard Turtle, W. Bruce Croft: Indri: A language-model based search engine for complex queries. Konferenciakiadvány, International Conference on Intelligence Analysis (2004).