Gépi ékezés

Kornai András
IBM Almaden Research Center
és
MTA Nyelvtudományi Intézet

Tóth Gábor
Sterrenkundig Instituut, Rijksuniversiteit Utrecht
és
ELTE TTK Atomfizika Tanszék

Cikkünkben a számítógépen tárolt illetve hálózaton továbbított magyar szövegekből gyakran hiányzó ékezetek pótlásának problémájával foglalkozunk. Módszerünk lényege, hogy összegyűjtünk minél több helyesen ékezett szóalakot, majd az ékezettelenített és az eredeti ékezett alakok közti kapcsolatot statisztikai feldolgozás után beépítjük a programba.

0 Bevezetés

Az ékezetes magyar betűk (és általában a latin ábécén alapuló nemzeti ábécék sajátos grafémáinak) számítógépes tárolása és hálózati továbbítása részben máig megoldatlan feladat. A probléma nem a nemzetközi szabványok hiánya, hanem éppen ellenkezőleg, az egymással versengő szabványok sokasága. Az átlagos számítógépfelhasználó nem tudja, hogy ugyanaz a karakter, pl. a rövid ö, más-más kóddal van eltárolva attól függően hogy DOS, Windows, Macintosh, vagy Unix PC-t használ, és megintcsak másképp az IBM illetve a többi nagyszámítógépen. Az átlagos felhasználó csak azt tudja, hogy különböző rendszerek között az ékezetek konvertálásával és hálózati továbbításával mindig baj van. Ezért a felhasználók igen nagy része óvakodik attól, hogy ékezetekkel írjon, különösen, ha a szöveget számítógépes hálózati terjesztésre szánja. Bár ez a korszerűbb gépek elterjedésével egyre inkább csökkenő probléma, azt is meg kell említenünk, hogy a régebbi PC-k és a nagyszámítógéphez kapcsolódó terminálok nem mindig képesek az ékezetes betűk megjelenítésére, illetve billentyűzetük ezek bevitelére alkalmatlan. Mindennek eredményeképp igen gyakran találkozunk olyan szövegekkel, mint az alábbi:

(1) `Potyka bacsit kituntettek, Pongratz Gergely liberalis tetuzott, Giczy szerint idegen kisebbseg uralkodik a kereszteny nemzeti tobbsegen, Szabo Albika Izraelbe telepitene a zsidokat. Vasarnap Jean Marie Le Pen fog szonokolni a Parlament elott. Gyulekezes a Koztarsasag teren.`

Természetesen minden magyarul írni-olvasni tudó ember tisztában van vele, hogy a fenti szöveg helyesen így nézne ki:

(1') `Potyka bácsit kitüntették, Pongrátz Gergely liberális tetűzött, Giczy szerint idegen kisebbség uralkodik a keresztény nemzeti többségen, Szabó Albika Izraelbe telepítené a zsidókat. Vasárnap Jean Marie Le Pen fog szónokolni a Parlament előtt. Gyülekezés a Köztársaság téren.`

Az ékezetpótlás problémáját az tette aktuálissá, hogy az interneten egyre inkább elterjedő WorldWideWeb lehetővé teszi az ékezetes betűk megjelenítését. Bevitelük változatlanul nem problémamentes, pl. az `és` szót a WWW alapját képező HTML nyelvben a `és` kifejezéssel kell lekódolni, de ez csak a kézzel gépelő embernek okoz nehézséget, a programoknak nem.

Cikkünk első részében a rendszer elvi alapjait írjuk le. Az algoritmussal a második részben, a rendszer korlátaival és bővítési lehetőségeivel pedig a harmadik részben foglalkozunk.

1 Szótári keresés

Míg az emberi nyelvi kompetenciát az ékezetek visszapótlása szinte meg sem terheli, addig a számítógépes nyelvészeti algoritmusok számára, legalábbis ezek jelenlegi fejlettségi szintjén, a feladat 100%-os vagy azt megközelítő megoldása egyszerűen lehetetlen. Vannak persze olyan esetek, amikor az ékezetes alak az ékezettelenből egyértelműen visszaállítható: pl. az ékezettelen `bacsi` összes lehetséges ékezetes változata `bacsi,bacsí,bácsi,bácsí` közül egy és csak egy szerepel a magyar nyelv szókészletében (melybe nemcsak a szótári alakot, hanem az összes morfológialiag jólformált szóalakot is beleértjük). De az esetek nagy részében a feladat ennél jóval bonyolultabb, pl. a `kituntettek` szó helyes ékezését csupán a tárgy határozottságának vizsgálatával (`kitüntettek egy embert` vs. `kitüntették az embert`) állapíthatjuk meg, ez pedig a mondat egészének szintaktikai elemzését igényli. Sőt olyan példát sem nehéz találnunk, ahol az elemzés tágabb értelemben vett szemantikai tényezőket, a világra vonatkozó enciklopédikus tudást is számításba kellene vegyen, így pl. az ékezettelen `baba` szónál:

(2a) `Nehéz szülés volt. A bába teljesen kimerült.`

(2b) `Nehéz szülés volt. A baba teljesen elkékült.`

Bár vannak olyan mesterséges intelligencia adatbázisok, amelyek éppen az ilyen "mindennapi" tudást igyekeznek modellezni (Lenat 1995), de gondoljuk csak végig, mi minden kellene ahhoz, hogy ezt a tudást a probléma megoldásához hasznosítani tudjuk. Kellene elsősorban egy olyan logikai következtetőrendszer (inference engine) amely axiómák tízmillióit képes áttekinteni – a jelenlegi szakértői rendszerek legfeljebb tízezer axióma mellett hatékonyak. Kellene továbbá egy olyan tudásreprezentációs (knowledge representation) formalizmus, amelynek formuláival a természetes nyelvek összes releváns jelentésárnyalata megkülönböztethető. Miután a (2a-b)-hez hasonló ékezetproblémát minden szemantikai problémához könnyű konstruálni, elvben az ékezetek visszapótlásához egy teljes szemantikára van szükség, annak felhaszálásához pedig egy teljes szintaxisra. A magyar (és általában az agglutinatív nyelvek) szintaktikai elemzése morfológiai elemzést is előfeltételez, és bár igaz, hogy erre viszonylag hatékony algoritmusaink vannak, de ezek mind felhasználják az ékezetekben rejlő információt, tehát erre a célra újra kellene írni a morfológiai elemzőket is.

Miután a probléma teljes megoldása előfeltételezi a számitógépes nyelvészet és a mesterséges intelligencia-kutatás számos, évtizedek óta nyitott problémájának megoldását, ezért nem szimbólum-kezelő szabályokon alapuló, hanem statisztikai megoldást keresünk. Természetesen a szimbólumok és szimbólum-sorozatok manipulálására így is szükség van, de mint látni fogjuk, a rendszer ereje nem a manipuláció logikai mélységéből, hanem a manipulálandó adatok bőségéből származik. Rendszerünk tehát nem a hagyományos "kemény mesterséges intelligencia" (hard AI) hanem a statisztikai alapú számítógépes nyelvészet irányzatába tartozik. Miután ez az irányzat hazánkban elég kevéssé ismert, külön kimondjuk az összes olyan előfeltevést is, amit a témában járatosabb olvasó esetleg triviálisnak érez.

1.1 Definíciók

Jelöljük azt a függvényt amely az ékezetes szövegből elhagyja az ékezeteket b-vel, b inverzét pedig C-vel. (Matematikai szemmel nézve C nem függvény, hanem reláció.) Legyen a magyar nyelv szókincse (ékezetekkel) V. Ha G a magyar grafémák halmaza $G = \{a,\ á,\ b, \ldots.x, y, z\}$ (külön grafémának számítjuk az ékezetes betűket, de a digráfokat és trigráfokat nem), akkor a hagyományos megközelítésben V mint a G elemeiből képzett láncok G^* halmazának egy részhalmaza adott. A továbbiakban egy *ékezettelen* x láncot *unikusnak* nevezünk, ha $C(x) \cap V$ üres vagy egyelemű halmaz, *veszélyesnek* ha többelemű. A `bacsi` tehát unikus, míg a `baba` veszélyes lánc.

A statisztikai elemzés első lépése az, hogy V-t nem úgy fogjuk fel, hogy G^* minden egyes eleméről egy 0-1 döntést testesít meg, hanem úgy, hogy minden egyes elem egy 0 és 1 közti valós számmal, a *gyakorisággal* van jellemezve. Külön felhívjuk a figyelmet arra, hogy egyes nyelvtanilag helytelen (agrammatikus) láncok szerepelhetnek pozitív gyakorisággal, és hogy nyelvtanilag kifogástalan láncok is szerepelhetnek 0 gyakorisággal, ha nem is a populáció egészében, de az egyes mintákban. Legyen P az egyes szavak relatív gyakoriságát (a populációban való előfordulásuk valószínűségét) megadó $G^* \rightarrow [0, 1]$ függvény. P-t nem ismerjük pontosan, de értékeit a populációból vett különféle minták azaz *korpuszok* alapján meg tudjuk becsülni. Például a félmillió szavas korpuszon alapuló Magyar Nyelv Gyakorisági

Szótára (Füredi-Kelemen 1989) alapján az `és` kötőszó gyakorisága 1.84%, míg a Magyar Narancsból vett hasonló méretű minta alapján az `és` gyakorisága 1.65%.

Mint a következő részben látni fogjuk, algoritmusunk alapja az, hogy a szavak ékezetes formáit tároljuk a program memóriájában. A veszélyes szavaknál kétféle módszert követhetünk: az *óvatos* algoritmus veszélyes szavak esetén jelzi a két- vagy többértelműséget, de nem választ az alternatívák között, míg a *bátor* algoritmus valamilyen kritérium alapján kiválaszt egyet, pl. a leggyakoribbat. A tároláson alapuló algoritmusok sikerességét előre meg tudjuk becsülni annak alapján, hogy a tárolt szavak összesített gyakorisága H mekkora. Egy óvatos algoritmus, amely a magyar szókincs H részét lefedő listán alapul, megközelítőleg az esetek (1-H)/2 részében fog hibázni (90%-os lefedettség esetén tehát 5%-ban), mert megközelítőleg a szavak fele eleve nem tartalmaz ékezetet.

Tekintsük most azt a bátor (de ügyetlen) algoritmust, amely veszély esetén a lehetséges formák közül mindig az ékezettelen alakot választja. Legyen a populációban az unikus illetve a veszélyes szavak valószinűsége u illetve v: definíció szerint $u + v = 1$. Miután a magyarban $u \approx 0.75, v \approx 0.25$, és nagyjából a szavak fele ékezettelen, ez az algoritmus a veszélyes szavak felénél fog hibázni, tehát legfeljebb 87.5%-os pontosságot érhet el, de még ez is csak akkor lehetséges, ha a memóriában minden unikus szó(alak) tárolva van. Mint a következő részben látni fogjuk, a szóalakok tárolásának elsődleges technikai korlátja nem a rendelkezésre álló memória mérete, hanem a tárolt szójegyzék szükségszerűen hiányos volta.

2 Az algoritmus

Programunk 5 egymást követő részből áll: előfeldolgozás, a gyakori szavak kikeresése, a rövid szavak kikeresése, a fennmaradó szavak kikeresése, utófeldolgozás. Vegyük ezeket sorra.

2.1 Előfeldolgozás

A bejövő szöveg általában az ékezetek hiányán túl is számos jelét viseli annak, hogy számítógépen írták és hálózaton továbbították. Igen gyakoriak benne az e-mail címek, pl. `kovacs@pluto.hu`, amiket nem kivánunk kijavítani a kétségtelenül igényesebb helyesírást tükröző `Kovács@Plútó.Hu` formára, mert az íly modon címzett e-mail továbbítására a hálózat jelenleg még nem képes. Hasonló módon, a WWW-n igen gyakori URL kifejezések (uniform resource locators) is benn kell maradjanak az eredeti (7-bites) ASCII szabványon belül.

Az előfeldolgozás tehát felismeri az ilyen kifejezéseket, és ezeket félreteszi. Félreteszi továbbá az írásjeleket, a számokat, és általában mindazokat a láncokat amelyek nem tartalmaznak magánhangzót.

A fennmaradó láncokat viszont kapcsos zárójelek közé teszi: a program további lépései már csak az íly módon megjelölt szavakat vizsgálják. Ha tehát a kiinduló szöveg az alábbi:

```
Felado : Magyar Narancs [Hungary]
Temakor: Tartalom ( 97 sor )
Idopont: Tue Jul  9 09:41:18 EDT 1996 NARANCS1 #73
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
VIII. evf. 28. szam, 1996. julius 11.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            http://www.hungary.com/narancs/8_28/joogyula.jpg
  Joo Gyulat abrazolja a fenykep. Talan az o hangjat
  lehetett a legkevesbe hallani, harsanyak szeretunk lenni, o
```

akkor az előfeldolgozás után az alábbi szöveget nyerjük:

```
{Felado} : {Magyar} {Narancs} [{Hungary}]
{Temakor}: {Tartalom} ( 97 {sor} )
{Idopont}: {Tue} {Jul}  9 09:41:18 {EDT} 1996 {NARANCS}1 #73
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
{VIII}. {evf}. 28. {szam}, 1996. {julius} 11.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            http://www.hungary.com/narancs/8_28/{joogyula}.jpg
  {Joo} {Gyulat} {abrazolja} {a} {fenykep}. {Talan} {az} {o} {hangjat}
  {lehetett} {a} {legkevesbe} {hallani}, {harsanyak} {szeretunk} {lenni}, {o}
```

Az előfeldolgozó reguláris kifejezéseken alapuló igen egyszerű és rendkivül hatékony program: az `UNIX` környezetben elérhető `lex` elemző segítségével generált program 18000 szót (75000 karakter) dolgoz fel egy másodperc alatt. Ugyanez a program a WWW alkalmazásokban igen elterjedt Perl nyelven megírva háromszor-négyszer lassabb (adataink egy IBM RS/6000 munkaállomásra vonatkoznak, melynek sebessége a ma átlagosnak tekinthető 100MHz Pentium PC-k sebességének durván kétszerese).

2.2 A gyakori szavak

Az a program, ami nem csinál semmit, az esetek felében (nagyobb korpuszon lemérve 54%-ban) "sikeres". Ha programunk semmi mást nem csinálna, csak a tíz leggyakoribb unikus szónál tenné ki az ékezeteket, tehát `es` helyett `és`, `ket` helyett `két`, stb., akkor ezzel a találati arány mintegy 3%-kal javulna. Egy némileg nagyobb, de elveiben semmivel sem bonyolultabb programunk ezt a cserét az ötezer leggyakoribb szóra végzi el, ezzel a találati arány 78%-ra javul.

Ennél is sokkal fontosabb hatás, hogy a gyakori szavak átnézésével a további programokra háruló feladatok erősen csökkennek. Az előfeldolgozás a szavak mintegy 8%-át vonja ki a további feldolgozás alól, míg a gyakori szavak kikeresése után mindössze az eredeti anyag 27%-ával kell foglalkozni. A `lex` elemzők hatékonyságát jól mutatja, hogy az ötezer gyakori szó átnézését 12000 szó/sec sebességgel végzi, tehát az előfeldolgozó sebességénél mindössze egy harmaddal lassabban. Perl-ben a program ismét egy hármas faktorral lassabb.

2.3 A rövid szavak

Algoritmusunk alapgondolatával összegyeztethető lenne, ha a további szótári keresést (jelenleg mintegy 150000 szóalakot tárolunk) is egy hasonló `lex` elemzővel nézetnénk át. Ennek azonban határt szab az elemzők tárigénye: míg az előfeldolgozó egészen parányi (12 kilobyte), a gyakori szavak tárigénye már két nagyságrenddel nagyobb (755 kilobyte), az egész szótárra kiterjedő hasonló program pedig több mint 20 megabyte (MB) lenne. Miután a program egy olyan PC-n fut, melynek jelenleg összesen 32MB memóriája van, és az ékezetek pótlása csak egyike annak a számos feladatnak amit ez a gép ellát, mindenképpen szükség van arra, hogy a rendszer méretét szigorú korlátok között tartsuk.

A 150000 szavas nagyszótárban a szavakat ezért tömörítve tároljuk, olyan formában, hogy együtt is csak 1.8MB memóriát igényeljenek. A keresés alapja a számítógéptudományból jól ismert hash-függvények felhasználása (Knuth 1988 vol 3 ch 6.4), melyről 2.4-ben még bővebben írunk: most csupán annyit jegyezzünk meg, hogy a hash egy olyan függvény amely minden betűsorhoz egy egész számot rendel. Ha pl. az `a` betűt 0-val, a `b`-t 1-gyel, a `z`-t pedig 25-tel számoljuk, akkor minden szó megfeleltethető egy 26-os számrendszerben felírt pozitív egész számnak. Mi ezt a hash-t az ékezettelen szavak első, harmadik, és ötödik betűje alapján számoljuk (azért nem az első három betű alapján, mert az abból nyert eloszlás kevésbé lenne egyenletes).

Miután az eljárás csak az ötbetűs vagy annál hosszabb láncokra alkalmazható, külön kell foglalkoznunk a négybetűs vagy annál rövidebb szavakkal. Ezek jelentős része természetesen már a gyakori szavak listáján is szerepelt, a rövid szavak szótárában tehát csupán a fennmaradó mintegy 1400 olyan szóval kell külön foglalkoznunk amely nem eleve ékezetmentes. Az ezek alapján generált program lényegében a gyakori szavakat ékező programmal megegyező sebességű és tárigényű. Futása után az eredeti szavak újabb 2%-a kerül ki a tovább vizsgálandó szavak köréből, tehát a nagyszótárban az eredeti anyag csupán 25%-át kell majd megnézni.

2.4 A nagyszótár

A nagyszótári keresés alapja tehát a hash-függvények felhasználása: minden szóhoz egy számot rendelünk, a szótárat pedig előre felvágjuk kisebb darabokra avagy *cellákra* úgy, hogy az azonos hash-értékű szavak (és csak azok) kerüljenek ugyanabba a cellába. Ahelyett, hogy a szótárban az összes szót átnéznénk, az első lépésben kiszámoljuk a keresett szó hash-értékét, a második lépésben pedig a szót már csak az ehhez az értékhez tartozó cellában keressük. A fentebb leírt hash segítségével minden ékezettelen alakhoz egy 0 és 17575 közti számot rendelünk. Ha a függvény eloszlása tökéletesen egyenletes lenne, akkor a nagyszótárt $26^3 = 17576$ darabra vágtuk volna, és az egyes darabokba átlag 150000/17576=8.53 alak esne. A valóságban persze a hash megoszlása nem tökéletesen egyenletes, egyes értékek (pl. az `xxx`-hez tartozó 16169) soha nem fordulnak elő, mások viszont annál gyakrabban (a leggyakrabban `mge`, 1132-szer). Összesen 7696 érték lép fel, tehát az átlagos cellaméret ≈ 19.4.

Érdemes megemlíteni, hogy az átlagos cellaméret nem egyezik meg a keresések várható munkaigényével, hiszen nem mindegy, hogy a gyakran keresett szavak a nagy (tehát sok belső keresést igénylő) vagy a kis cellákba esnek. Ezért a továbbiakban nem az egyes cellák méretét, hanem ezek az odakerült szavak összgyakorisága szerint súlyozott átlagát, az ú.n. *várható ütközésszámot* fogjuk tekinteni: a fentebb leírt (az első, harmadik, és ötödik betűn alapuló) séma esetén ez ≈ 147. Tehát a véletlenszerűen (a szöveg gyakorisági megoszlását követő módon) érkező új szavak közül azok, amik eljutnak a nagyszótárig (mert sem a gyakori sem a rövid szavak szótárában nem találtuk meg őket) átlagban 147/2 másik szóval kell összehasonlítani (ütköztetni) ha a cellán belül lineáris keresést alkalmazunk, illetve $\log(147)/\log(2) = 7.2$ szóval ha bináris keresést alkalmazunk.

Miután nagyszótári program egyes darabjainak lemérése világosan mutatja, hogy a program futásidejének javát ilyen összehasonlításokkal tölti el, a program gyorsításának legfontosabb eszköze a várható ütközésszám csökkentése. Ha az eddig figyelmen kívül hagyott második és negyedik betűt is beszámoljuk, akkor a várható ütközésszám lecsökken ≈ 54-re. Ennek azonban az lenne az ára, hogy a hash maximális értéke ne 17575 ($26^3 - 1$) hanem 11881376 ($26^5 - 1$) legyen, ami bináris alakban 24 bitet igényel. Technikai okokból a 24 bit (3 byte) nem előnyös, tehát megvizsgáltuk mi a helyzet 16 illetve 32 bit felhasználása esetén. Azt találtuk, hogy az 5 betűből kihozható optimumnál mindössze 10%-kal rosszabb eredmény, ≈ 59.5-ös ütközésszám elérhető már 16 bitben is, míg 32 bit lehetővé teszi, hogy ez első öt helyett az első hét betűt vegyük figyelembe, ami által a várható ütközésszám ≈ 14-re csökken. Összességében tehát a program szívét jelentő keresési algoritmus az első változathoz képest tízszeresen gyorsítható fel ha lineáris, és kétszeresen ha az eleve gyorsabb (de körülményesebben programozható) bináris keresést használjuk.

Szerencsés esetben a szó kikeresésével a nagyszótári keresés már véget is ért: ha megtalájuk a szót, akkor a szóval együtt tárolt ékezetminta alapján kipótoljuk az ékezeteket, eltávolítjuk a szövegbeni keresést vezérlő { } jeleket, és már készen is vagyunk. Ha a szót nem találjuk, akkor további *indirekt* keresési módszereket alkalmazunk, ezekkel a 3.2 részben foglalkozunk.

2.5 Utófeldolgozás

Az utófeldolgozást végző program két feladatot lát el: egyrészt törli a nagyszótári keresés sikertelenségére utaló { } jeleket, másrészt kijavít néhány olyan sablonhibát, amit az okoz, hogy a szavakat a központozástól elválasztva vizsgáljuk. Gyakorisági megfontolások alapján teljesen egyértelmű, hogy a különálló o szót ő-re kell cserélni. Ha viszont az o két pont között szerepelt, akkor az `oldal` szó rövidítése, és mint ilyen változatlanul kellene maradjon. De mível az előfeldolgozó program a `.o.` láncot `.{o}.` formára hozza, és mível a főprogram csak a { és } közti részt vizsgálja, az utófeldolgozóra marad a `.ő.` lánc visszaállítása a helyes `.o.` formára. Hasonló ehhez a `-es` lánc viselkedése: kötőjel után ez kizárólag az `MDF-es, 1971-es` stb. láncok részeként szerepel, de a gyakori szavak szótára után ezek a hibás `MDF-és, 1971-és` formában szerepelne, ha az utófeldolgozó nem javítaná. Az utófeldolgozó az előfeldolgozóhoz hasonlóan kis méretű (lex-ben és Perl-ben mindössze 5 sorból álló) és nagyon gyors program.

3 A statisztikai módszer korlátai

Természetesen várhattunk volna arra, hogy a korszerűbb számítógépek és a kelet-európai ékezetszabvány (ISO 8859-2) elterjedésével az emberek előbb-utóbb felhagynak az ékezetek nélküli gépelés rossz szokásával, és várhattunk volna arra is, hogy a számítógépes morfológia, szintaxis, szemantika, diskurzus-elmélet, tudásreprezentáció, és tételbizonyítás összes problémáját pillanatok alatt megoldják. De úgy véltük, hogy hasznosabb egy korlátozott, de működő rendszert létrehoznunk.

Kétségtelenül van valami csábító abban, hogy a fenti tudományterületek mélyebb problémáit szinte teljesen ki tudjuk kerülni egy direkt, a programozástechnikai részletektől eltekintve igen egyszerű, pusztán memorizáláson alapuló séma felhasználásával. Úgy véljük, hogy eljárásunk a nyelvtudományban megszokott, egyedi példákon alapuló érveléssel nem is támadható mindaddig, amíg valaki legalább ugyanilyen sikeres, de statisztika helyett szabályokon alapuló rendszert nem produkál.

Teljesen jogos azonban a kérdés, hogy vajon tisztán statisztikai szemszögből mennyire tekinthetjük sikeresnek a rendszert. Az erre adott válasz részben a felhasználók céljaitól függ: 3.1-ben tehát először ezzel foglakozunk. 3.2-ben és 3.3-ban azt írjuk le, hogy a következetes statisztikai elemzés milyen formában hozza mégis elő a hagyományos nyelvészet, különösen a morfológia és a szintaxis problémáit, végül 3.4-ban levonjuk a kínálkozó tanulságokat.

3.1 A hibák mérése

Azt, hogy mit tekintünk hibának, elsősorban a felhasználó végcélja szabja meg. Ha célunk a szöveg olvasása, akkor az óvatos rendszer által bennhagyott ˇ ékezetjelek zavaróak, valószínűleg szubjektíve sokkal inkább zavaróak mint a kihagyott ékezetek. Ha viszont célunk a szöveg helyes változatának előállítása, akkor a ˇ hasznos, hiszen vezeti a szemet, míg a kihagyott ékezet után keresgélni kell. A hibák súlyát a következő módon számoljuk:

	a	á	e	é	i	í	o	ó	u	ú	ö	ü	ő	ű
	0	0.4	0	0.4	0	0.2	0	0.4	0	0.3	0.5	0.5	0.5	0.5
´	0.6	0	0.6	0	0.3	0	0.6	0	0.4	0	1	1	1	1
¨	-	-	-	-	-	-	0.9	1	0.9	1	0	0	0.3	0.3
˝	-	-	-	-	-	-	1	1	1	1	0.4	0.4	0	0
ˇ	0.6	0.4	0.6	0.4	0.3	0.2	0.6	0.4	0.4	0.3	0.4	0.4	0.3	0.3

Bár az egyes hibáknak tulajdonított numerikus értékek némileg önkényesek, táblázatunk úgy véljük helyesen tükrözi az átlagos olvasó preferenciáit:

- A legzavaróbb hibák az o/ó és u/ú helyetti ő illetve ű, továbbá az ö/ő és ü/ű helyetti ó illetve ú.
- A legkevésbé zavaró hiba az í helyetti i.
- Ékezet hozzáadása zavaróbb mint ékezet elhagyása.
- Azok a hibák, amelyek csak a magánhangzó hosszát befolyásolják, kevésbé zavaróak mint azok, melyek minőségi különbséget is jelentenek.
- A ˇ ékezet kitétele legfeljebb ugyanakkora hiba mint az adott oszlop legkisebb hibája (óvatos módban ennek is csak a felét indokolt felszámítani).

Igényesebb felhasználó-barátsági (user-friendliness) illetve pszicholingvisztikai vizsgálatok ugyan még módosíthatják némileg a fenti táblázatot, de az már jelen formájában megfelel annak a célnak, hogy a hibás alakok mechanikus leszámolása helyett egy finomabb statisztikai mérőszámot adjon. Egy tipikus elektronikus cikken (a Batthyány Lajos Alapítvány Napi Sajtószemléjenk október 29-i számán) összehasonlítva a kettőt, azt találjuk, hogy a program annak 2280 szavából 95-ben nem találja meg a helyes ékezetet, ami 4.16%-os hibaarány, viszont fenti táblázat értékeivel számolva 5539 magánhangzón 87.1 hibapontot kapunk, ami mindössze 1.58%-os hibaarány.

Érdemes megemlíteni, hogy a hibák nagyjából fele olyan, hogy a kontextus ismerete nélkül nem javítható: `szerepĕt,vezettĕk` stb. Bár az ilyesfajta ragozási többértelműség előfordul mély hangrendű szavaknál is (pl. `megrongáltăk`) az `é/e` hibák több mint kétszer olyan gyakoriak mint az `á/a` hibák. Akár a táblázat szerint súlyozva számoljuk akár darabszámra, az `é/e` és `á/a` hibák adják az összhiba mintegy kétharmad részét.

3.2 Morfológia és indirekt keresés

Mint az előbbi szakasz példái is mutatják, igen gyakran találunk olyan hibákat, amiket még a helyes morfológiai elemzés ismeretében sem tudunk kijavítani – olyan gyakran, hogy ez a morfológiai elemzés hasznosságát eleve kérdésessé teszi, különösen annak a fényében, hogy statisztikai alapon épített morfológiai elemzőhöz konzervatív becslés szerint is legalább harmincmilliárd szóalakra (tehát a jelenleg elektronikus formában elérhető szövegeknél négy nagyságrenddel többre) lenne szükség (Kornai 1992). Ugyanakkor az is igaz, hogy a szavak többsége a mintában csak egy- vagy kétféle ragozott alakban fordul elő, tehát a rendszernek képesnek kell lennie eddig még nem látott ragozott formák kikövetkeztetésére, a szótő alapján való *indirekt* keresésre is.

Rendszerünk ezt egy hatékony, bár nyelvészeti szempontból önkényes algoritmussal oldja meg. Ha pl. a `baranyborbe` szóalakot nem találjuk meg a szótárban, akkor sorra megnézzük az `baranyborb`, `baranybor`, `baranybo`, `baranyb` kezdetű alakokat (jelenleg legfeljebb 4 betű levágását engedjük meg, és megköveteljük, hogy a maradék legalább 5 betű hosszú legyen). A példában ez az eljárás az első lépésben sikerre vezet, ugyanis a `baranyborbol` alak szerepel a nagyszotárban, és tudjuk hogy ennek a helyes ékezése `báránybőrből`. Ebből tehát a keresett darab `báránybőrb-` nek adódik, és ehhez még hozzátesszük a (mindig ékezettelenül hagyott) `e` végződést, ami a helyes `báránybőrbe` formát adja. Természetesen az eljárás sikere nem garantált, pl. az `elszaporodasatol` forma esetén az algoritmus az `elszaporodása` formát találja meg, és az ebből mechanikusan visszállított `elszaporodásatol` nem tükrözi sem a `-tól` inherens hosszú ó-ját, sem azt, hogy nyílt-magánhangzó-nyúlás (Nádasdy-Siptár 1994:2.2.3.1) hatására a birtokrag `a`-ja helyett `á`-t találunk.

A módszer a nagyszótárban nem talált szavak harmadát találja meg, és ezeknek kicsit több, mint felén ad hibátlan eredményt. De ahol nem hibátlan, ott is számos ékezetet javít: alkalmazása nélkül a tesztelt anyagon az összesített hibapontszám nem 87.1, hanem 106 lenne. A toldalékok inherens ékezeteinek beépítése után a hibapontszám 78.5-re csökken, a nyílt-magánhangzó-nyúlás figyelembevétele pedig ezt 73.1-re javítja. Becslésünk szerint ezzel a két kiegészítéssel a mechanikus vágó-algoritmus a teljes morfológiai elemzéssel kijavítható hibák több mint 80%-át kijavítja. A jelenleg a `www.hungary.org/ekito.cgi` alatt futó rendszerben, amely a `cs.rice.edu` `public/andras` könyvtárában `ekesit.tar.gz` néven érhető el `ftp`-vel, ezek a kiegészítések még nem szerepelnek.

3.3 Kitekintés a szintaxisra

Rendszerünk jelenleg még nem tartalmaz szintaktikai elemzőt, tehát az ebben a szakaszban foglaltak némileg spekulatív jellegűek. Mint említettük, a rendszer által nem javított ékezethibák mintegy fele csak a kontextus ismeretében lenne biztonsággal javítható. Az ilyen hibák 40%-a a szófaj ismeretében már javítható lenne: pl. `mĕg, ŭgy, szŏba`. A statisztikai nyelvészet módszereivel a szófaj meghatározása (tagging) igen nagy sikerrel megoldható, sőt csak így oldható meg: az u.n. rejtett Markov-láncokon alapuló szófajmeghatározó programok hatékonysága messze felülmúlja a hagyományos szabály-alapú szintaktikai szófajmeghatározók hatékonyságát.

Még két nagy hibaosztályt találunk: az alanyi és tárgyas ragozású igék (mintánkban pl. `vettĕk`) illetve a birtokjeles illetve jelöletlen esetragos formák (`hatalmăt, küldöttgyűlésĕn`) a szintaktikailag javítható hibák újabb 30-30%-át jelentik. Mintánkban mindössze egy olyan szintaktikailag javítható hiba volt ami a fenti három osztály egyikébe sem sorolható, ez a `korú/körű` pár, melynek feloldásához az egyes elemek szelekciós megkötéseit is figyelembe kellene venni. (Lesznek olyanok, akik ezt nem is a szintaxis, hanem a szemantika körébe utalják). Bár magyar változat még tudtunkkal nem létezik, a főnévi csoportok megtalálásában (chunking) is hatékonyabbak a statisztikai módszerek a szabály-alapú rendszerereknél.

Összességében arra számíthatunk, hogy statisztikai alapú szintaktikai elemzéssel a rendszer hibája a jelenlegi felére csökkenthető, és ennek a javulásnak a kétharmada már a legegyszerűbb, a szavak helyett a szópárok gyakoriságát számontartó u.n. bigramma-elemzéssel is elérhető. Figyelembe kell venni azonban azt is, hogy a szintaktikai elemzés a rendszer tárigényét egy nagyságrenddel megnövelné, sebességét pedig hasonló mértékben csökkentené.

3.4 Tanulságok

Programunk első és legfontosabb tanulsága az, hogy a gyakorlati problémák megoldásának nem előfeltétele az elméleti problémák megoldása. Bár a mérnöki elme számára evidens, hogy a könnyebb feladatot nem érdemes a nehezebbre visszavezetni, a tudományról vallott felfogást erősen áthatja az a "felülről vezérelt" (top down) modell, melyben az elméleti eredményeket az alkalmazások csak akadozva, gyakran jelentős lemaradással követik. Meglehet, hogy a matematika és a fizika, illetve a fizika és a fizikán alapuló technológia esetében valóban ez a helyes modell. De az elméleti megismeréstan (cognitive science) és a mesterséges intelligencia nincsenek abban a helyzetben, hogy a nyelvészeti jellegű alkalmazásoknak utat mutassanak, hiszen kulcsproblémáik nagyrészt megoldatlanok.

Ugyan kevésbé éles formában, de hasonló tanulság vonható le a szimbólum-manipuláción alapuló nyelvészet és a statisztikai elemzés viszonyáról is. Látjuk, hogy a megfelelő eszközökkel az elméleti nyelvészetben régóta számon tartott jelenségek (pl. a toldalékok inherens és kontextuálisan meghatározott tulajdonságainak elkülönítése, vagy a nyílt-magánhangzó-nyúlás törvénye) statisztikai alapon is relevánsnak bizonyulnak. De látjuk azt is, hogy ezek a jelenségek csak akkor válnak észrevehetővé, ha az elméletileg esetleg érdektelen, de statisztikailag domináns jelenségkörről már sikerrel számot adtunk.

Ha valamire megtanított bennünket az utolsó harminc év megismeréstudománya, akkor ez az a felismerés, hogy az emberi memória "olcsó", a gépi memóriánál sokkal nagyobb bőségben rendelkezésre álló erőforrás, az emberi szimbólum-manipuláció pedig "drága", a számítógépes aritmetikai és logikai műveletekhez képest lassú és megbízhatatlan tevékenység. Amikor tehát egy olyan folyamat gépi modellezésével foglalkozunk mint az ékezés, melyet az emberek könnyen és hatékonyan végeznek, akkor érdemes olyan erőforrásokat használnunk amelyek az embereknek bőséggel rendelkezésükre állnak, és érdemes minimalizálni azokat a lépéseket, amelyek elvégzése az embereknek nehézséget okoz. Mindennek alapján tehát azt jósoljuk, hogy a szimbólum-manipuláción alapuló, "kemény mesterséges intelligencia" jellegű nyelvészetet a statisztikai módszereken alapuló elemzés egyre jobban háttérbe fogja szorítani, nemcsak az alkalmazott, hanem az elméleti kutatások körében is.

Felhasznált irodalom

Füredi Mihály és Kelemen József (szerk): *A mai magyar nyelv szépprózai gyakorisági szótára* Akadémiai Kiadó, Budapest 1989

Donald E. Knuth: *A számítógép-programozás művészete.* Műszaki Könyvkiadó, Budapest 1988

Kornai András: Frequency in morphology. In: Kenesei István (szerk): *Approaches to Hungarian* **IV** (1992) 246-268

Douglas B. Lenat: CYC: a large-scale investment in knowledge infrastructure *Communications of the ACM* **38** (1995) 32-38

Nádasdy Ádám és Siptár Péter: A magánhangzók. In: Kiefer Ferenc (szerk): *Strukturális Magyar Nyelvtan 2. Fonológia* 42-181 Akadémiai Kiadó, Budapest 1994