Multidiszciplináris tudományok, 9. kötet. (2019) 4 sz. pp. 348-355 **https://doi.org/10.35925/j.multi.2019.4.34**

AZ ÁTMENETALAPÚ ÉS GRÁF-TRANSZFORMÁCIÓS MÓDSZEREK ÖSSZEVETÉSE A MONDATOK FÜGGŐSÉGI ELEMZÉSÉNÉL

Csépányi-Fürjes László
szoftvermérnök, EFFCOM Informatikai és Szolgáltató Betéti Társaság
3529 Miskolc, Aulich L út 13, e-mail: csf.laszlo@gmail.com

Kovács László
egyetemi tanár, Miskolci Egyetem, Informatikai Intézet
3515 Miskolc, Miskolc-Egyetemváros, e-mail: kovacs@iit.uni-miskolc.hu

Absztrakt

A függőségi nyelvtan kitüntetett szerepet játszik a magyar nyelvű szövegek mondatelemzésében. A függőségi gráfok előállítására két fő módszer terjedt el a szakirodalomban. Az egyik irányzat az átmenetalapú (transition-based) szerkezet generálás, a másik megközelítés a gráf-alapú (graph-based) módszerek családja. Jelen cikkben ezen két módszer összevetését végezzük el saját implementációkon keresztül. Az összevetéshez a szabvány UD (Universal Dependency) honlapon található adatmintákat használtuk fel. A tapasztalatok megerősítik azt, hogy az átmenetalapú megközelítésnél lényeges szerepet játszik, hogy projektív vagy nem-projektív nyelvtanokra készült-e az algoritmus. Másrészt az eredmények azt is jól mutatják, hogy gráf-transzformációs szabály alapú megközelítéseknél könnyebb egy átlagos jósági szintig eljutni, de nehezebb az összes egyedi eset lefedése.

Kulcsszavak: *számítógépes nyelvészet, függőségi nyelvtan, átmenetalapú mószer, gráf transzformációs módszer*

Abstract

Dependency grammar is an important tool in semantical analysis of Hungarian text sources. There are two main approaches in the literature for construction of dependency graphs. One approach is the family of transition-based methods, while the other model uses direct graph construction operations starting from an initial dependency graph. In this paper, we compare these two methods on text sources in Hungarian. We present here our proposed methods for the implementation of the approaches. For the test evaluation, we use the sentence bank of UD (Universal Dependency) homepage. Our experiences show that the accuracy depends significantly from the applied method whether it supports only projective grammars or also non-projective structures are involved. Regarding the rule-based graph transformation approach, we could get initially a very good progress, but later it was very time-consuming to cover all special features of the language.

Keywords: *computational linguistis, dependency grammar, transition-based method, graph transformation method*

1. Bevezetés

A számítógépes nyelvészet egyik fontos területe a mondatok szerkezetének elemzése, a szintaktikai elemzés. A mondatszerkezet leírására igen gazdag modellkészlet jött létre, hiszen a különböző természetes nyelvek eltérő logikával és formalizmussal rendelkeznek, a természetes nyelvi jelenségeket különböző nyelvtan reprezentációs módszerekkel is le lehet írni. A létrejött nyelvtancsaládok között legjelentősebb a konstituensfa alapú leírások (phrase-structure grammar, constituency grammars) [1] családja, melyben a mondat szavait a mondat alapvető strukturális alany (NOUN-phrase) – állítmány (VERB-phrase) szerkezete szerint csoportosítjuk. Ez a fajta csoportosítás különösen a kötött szórendű nyelveknél eredményez egy jól áttekinthető szerkezeti leírást. A másik fontos nyelvtancsalád a függőségifa nyelvtanok (dependency grammar) köre, melynek gyökerei Frege algebrai logikájáig [2] nyúlnak vissza. Frege felfogásában a mondat jelentése a predikátumhoz köthető, melynek szemantikáját az argumentumai pontosítják. Ezen megközelítésben az argumentumot jelentő szavak függnek a predikátum egységtől, jelentésük és létük hozzá kötődik. Ezen modellnek a nyelvészetben történő közvetlen úttörő alkalmazása Tesnière [3] nevéhez kapcsolódik.

A függőségi nyelvtan [4] a mondat nyelvtani leírását a szavak közötti függőségi relációkon keresztül adja meg, ahol az egyes függőségi kapcsolatoknak más és más osztálya, jelentése lehet. A függőségben az egyik tag a domináns tag (head) és a másik szó a függő tag. A függőségi kapcsolat egy szülő-gyerek (PCR) kapcsolatnak felel meg, minden szóhoz maximum egy domináns szó rendelhető. A gyerekelemek maximális darabszámát a szó vegyértékének (valency) nevezik. A mondatelemzés során a szavakhoz kategóriákat rendelünk, melyek között meghatározzuk a függőségi viszonyokat. Ha az így létrejött függőségi fa leveleinek sorrendje megegyezik a hozzájuk tartozó szavak mondatbeli sorrendjével, akkor egy projektív megfeleltetés jött létre, azaz ekkor a függőségi nyilak nem keresztezik egymást.

A függőségi gráfok előállítására két fő módszer terjedt el a szakirodalomban. Az egyik irányzat az átmenetalapú (transition-based) szerkezet generálás, a másik megközelítés a gráf-alapú (graph-based) módszerek családja. Az átmenetalapú módszer a mondat szavainak szekvenciális feldolgozásával, az előfordulási kontextus feltárásával határozza meg a szavak közötti függőségi relációt. A kapcsolatok és a kontextus leírás meghatározásánál egy adott pontban több lehetséges lépés is megtehető, amelyek közül rendszerint egy osztályozási algoritmussal választják ki a nyertes lépést [5].

2. Átmenetalapú függőségi elemzés

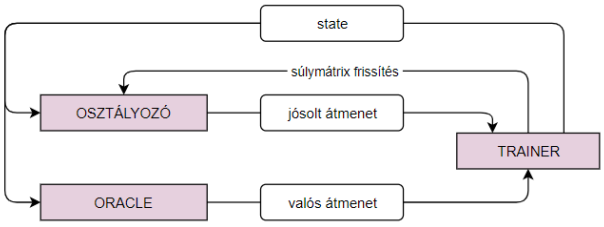
A nemzetközi irodalom igen gazdag a függőségi gráfot átmenetalapú megközelítéssel felépítő módszerekben. Az átmenetalapú módszerek [6] az alábbi adatelemekkel dolgoznak: állapotleíró konfigáció-vektor, induló konfiguráció és záró konfiguráció. Ezekhez kapcsolódnak az átmenet operátorok, melyek a konfigurációk halmazán értelmezettek. Az algoritmus az átmenet operátorok megfelelő sorozatával állítja elő a függőségi gráfot. Az arc-eager módszer az egyik legalapvetőbb és legelterjedtebb átmenetalapú algoritmus [7]. Az arc-eagar módszernél a konfiguráció-vektor három fő tagból áll: a még nem vizsgált szavak buffere; verem a részlegesen már vizsgált szavakhoz és a feltárt függőségek halmaza. A függőségi kapcsolatot a (domináns tag pozíciója, függő tag pozíciója; függőségi címke) hármas írja le. A modell az alábbi négy operációt tartalmazza: Local-Left, Local-Right, Reduce, Shift. A Shift operátor a bufferből áttesz egy szót a verembe, míg a Reduce a verem tetejéről vesz el egy szót.

Annak érdekében, hogy kontrollálni tudjuk az algoritmusok működését, illetve hogy hozzáférjünk a tulajdonság vektorok kezeléséhez, úgy döntöttünk, hogy elkészítjük a saját függőségi elemző implementációnkat. A rendkívül nagy számban szabadon elérhető könyvtárak miatt esett a választásunk a Java programozási nyelvre. Az implementáció kiinduló struktúráját a nyílt forráskódú Teraparser1 ihlette. A tesztek végrehajtásához az OpenJDK 11-es verziójú futási környezetet használtuk.

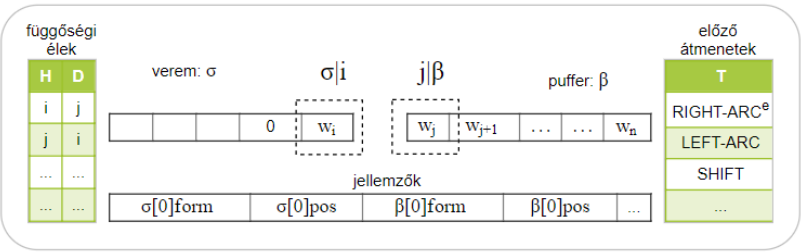
A tesztek inicializációs fázisában töltjük be a korpuszokat a memóriába. Kétféleképpen is tároljuk ezeket az adatokat. Az egyik módszer szerint az eredeti tanító, validáló, illetve tesztelő korpuszokat külön-külön, tömbökben tároljuk, tanítás után a teszt és a validáló korpuszokkal ellenőrizzük az eredményt. A másik módszer szerint mindhárom korpuszt összeolvasztjuk, az eredmény tömb mondatainak sorrendjét összekeverjük, majd 80%-20% arányban tanító és teszt halmazra osztjuk szét. Teszteléshez a címkézetlen kalkulált függőségeket hasonlítjuk össze a teszt halmazban lévő függőségekkel és százalékos formában jelenítjük meg az eredményt (UAS).

A korpuszok betöltése után példányosítjuk az oracle és a perceptron komponenseket, illetve magát a trainer szolgáltatást is ekkor paraméterezzük. Az implementációnkban a trainer kétféle frissítési stratégiával taníthatja a perceptron súlymátrixát. Ezek az alap-frissítési-stratégia és a pontszám-elosztó-stratégia.

A tesztelési fázisban a parser modult használjuk, hogy a betanított osztályzó perceptron segítségével előállítsuk a függőségi éleket, hasonló módon ahogyan a tanulási fázisban is történik. Maga a teszt összehasonlítja az előre definiált éleket és a kalkulált éleket egymással és kiszámítja a címkézetlen kapcsolatok pontszámát (UAS). Ezen értéket meghatározzuk mind a validációs mind a teszt korpuszra, valamint kiszámítjuk a tanító mintára is. Ezen százalékok segítségével osztályozni tudjuk a különböző konfigurációk hatását az algoritmusok működésének pontosságára.



1. ábra Iteratív tanulási fázis



2. ábra Verem alapú algoritmusok állapot objektuma

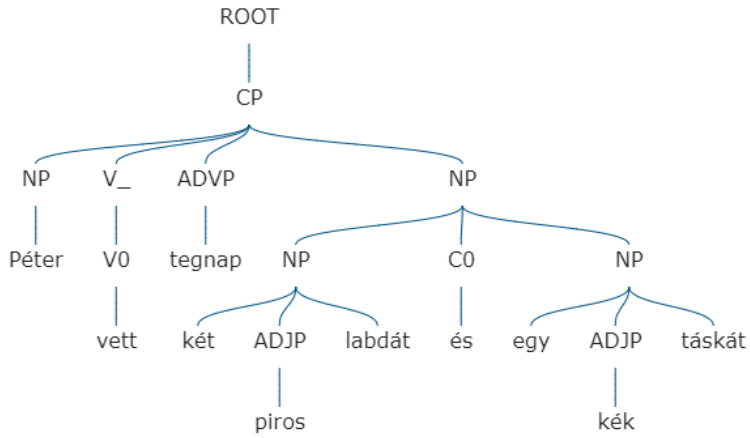
A verem-alapú változatoknak az állapot objektuma tartalmaz egy verem σ és egy input puffer β objektumot, valamint a már kikalkulált függőségi élek halmazát, az átmenetek listáját, illetve az állapot-jellemzőket vagyis a tulajdonság vektort (lásd 2. ábra). A lista-alapú változat állapot objektuma ettől

némileg eltér. Ebben a változatban két listában tároljuk azokat a tokeneket, amelyeket az algoritmus még nem dolgozott föl teljes mértékben. A jobboldali listafejjel rendelkező lista a fő lista λ1, míg a baloldali listafejjel rendelkező az átmeneti lista λ2. Ez az állapot objektum szintén tartalmaz egy puffert β, a már kikalkulált függőségi élek halmazát, az átmenetek listáját, illetve szintén tartalmazza az állapot-jellemző tulajdonság vektort.

Minden iterációban újabb állapot objektumot hozunk létre az azt megelőző állapotból, a korábbiakat pedig archiváljuk. Addig zajlik ez a folyamat, amíg el nem érjük a vég-állapotot, ami azt jelenti, hogy az input puffer β már üres és nincs token, ami feldolgozásra várna.

3. Gráf-transzformációs módszerek

A klasszikus függőségi gráfok fa struktúrát alkotnak, melynek egy absztrakt ROOT gyökere van és levelei az egyes szavak a vizsgált mondatban. A 3. ábra a „Péter vett tegnap két piros labdát és egy kék táskát” mondathoz tartozó függőségi gráfot szemlélteti. A gráf egyes részfái a mondat különböző mondatrészeit írják le.



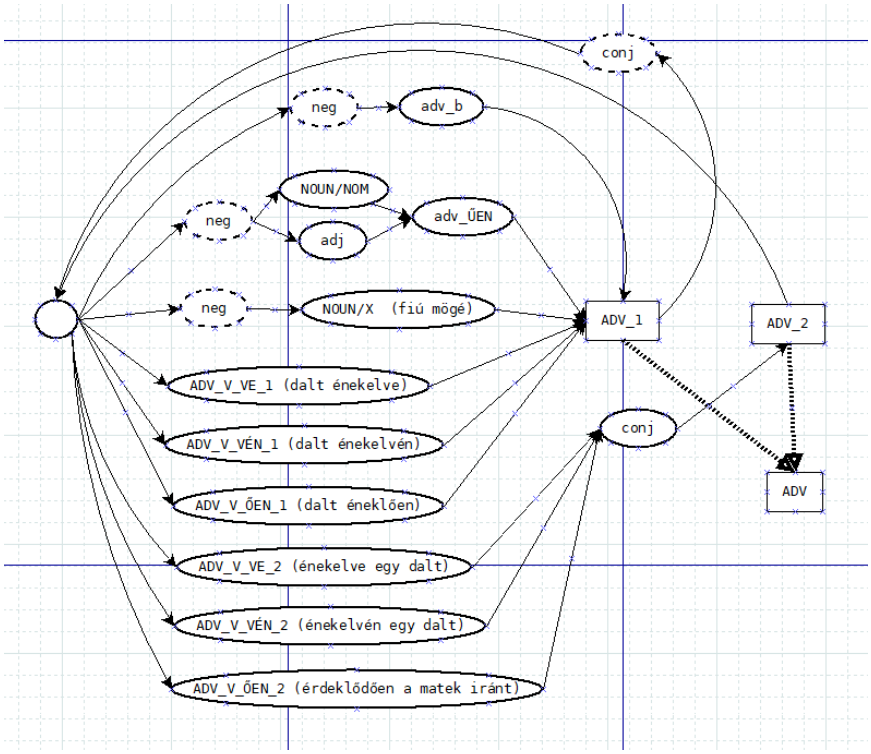
3. ábra Minta függőségi gráf [forrás: e-magyar.hu]

A gráf-transzformációs módszerek alapelve, hogy kiindulásként egy közelítő gráf struktúrát építenek fel, majd ezt a struktúrát lépésenként finomítják a faszerkezet átalakításával. A gráf-transzformációs módszerek gyökerei a 1990-es évek közepéig vezethetőek vissza. Ehhez kapcsolódóan kiemelhető Eisner munkássága [8], aki több generatív függőségi fa konstrukciós módszert is kidolgozott. A 2000-es évek első évtizedének közepére már megjelentek a tanuló algoritmusok elvén működő konstrukciós módszerek is. A kapcsolódó munkák közül megemlíthető McDonald, Crammer and Pereira két-fázisú konstrukciós algoritmusa [9], amelyben lefektették a leggyakoribb tanuló algoritmus, az arc-factoring módszer alapjait. Ezen módszer lényeges eleme, hogy az egyes élekhez egy súlyérték rendelődik. Az él ezen súlyértéke egyfajta valószínűséget jelöl, és cél a nagyobb súlyú élekből álló gráfok meghatározása. A gráf jósága a tartalmazott élekre vonatkozó súlyok összértéke lesz. Az arc-factoring alapú optimalizáció az optimális fa megkeresését a kifeszítőfák meghatározási elvén

működő algoritmusokkal végzi el. Ezen módszer egy többlépcsős, összetett optimalizációs, tanuló eljárás révén jut el az eredmény fához. Ezen megközelítésnél nehézséget jelent a tanító minta meghatározása és a módszerek optimális paramétereinek kijelölése [10].

Ezen problémák kikerülésére mintarendszerünkben nem egy tisztán gráf-transzformációs módszert alkalmaztunk, hanem a szabály alapú megközelítéssel kombináltuk az említett gráf-transzformációs módszert. A szabály alapú megközelítések gyökerei a 1960-es évekig nyúlnak vissza. Az első eredmények között megemlíthető Gaifman dolgozata [11], melyben bemutatásra került, hogy a projektív nyelvi elemek leírhatóak CFG, környezetfüggetlen nyelvtanokkal. A szabály alapú módszerek előnye, hogy tömören le lehet írni vele a szakértők tudását és tapasztalatait. Nem igényel külön betanítási fázist, determinisztikusan működik. Elsősorban az egyszerűbb, általánosabb nyelvi elemeket lehet a segítségével hatékonyan reprezentálni. Összetettebb esetekben azonban már elveszíti hatékonyságát, egyre költségesebb lesz megfelelő pontosságú szabályrendszert létrehozni.

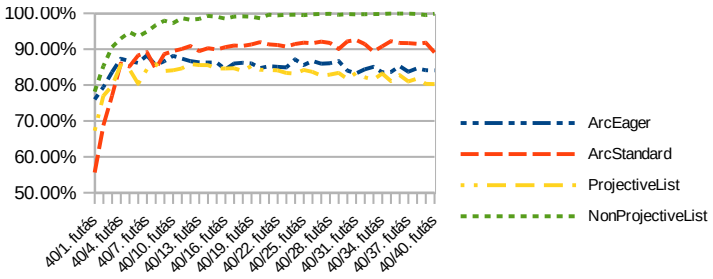
Rendszerünkben gráf-séma alapú szabályrendszert dolgoztunk ki. A létrehozott szabályok antecedens részei gráf mintákat jelölnek ki. A konzekvencia rész a mintarész összefoglaló szimbólumát adja meg. Egy-egy szabály egy-egy helyettesítési mintának is tekinthető, azaz, ha a megadott séma megtalálható a gráfban, akkor a kijelölt részt összefogjuk egy absztraktabb egységbe. A konténer rész tartalmazni fogja a kijelölt gráf részletet. Egy egyszintű, flat, gráfból kiindulva a helyettesítések láncolatán keresztül egy összetett függőségi gráf építhető fel.



4. ábra Minta a gráf séma szabályrendszerre

4. Teszteredmények

Első tesztünk az átmenetalapú függőségi elemzés algoritmus tanulási képességét vizsgálta a tanulási fázis ismétléseinek függvényében. Tesztünkben az alap-frissítési-stratégiát alkalmaztuk és azt vizsgáltuk, hogy a tanulási iterációk számának emelésével, magán a tanító mintán végzett teszt mennyire tudja megközelíteni a 100%-ot. A kapott eredmény közvetve használható a nem-projektív valamint a projektív algoritmusok magyar nyelvű mintát használó tanulásának eredményességi összevetésére.



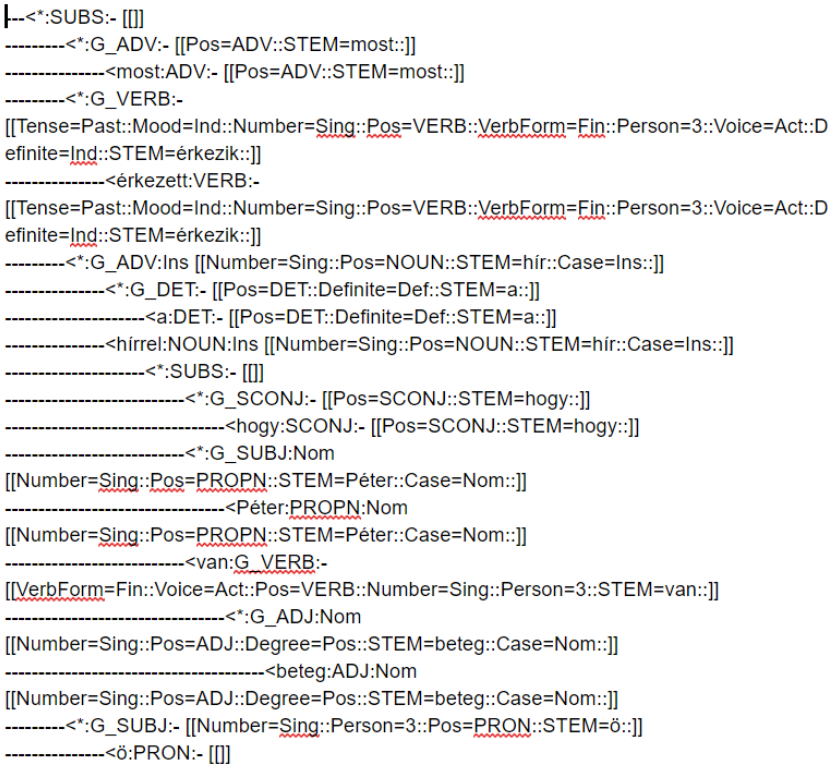
5. ábra Tanulási képesség mérése magyar tanítómintán

Mivel a teszteléshez jelen kísérletben magát a tanítómintát használtuk és nem egy szokásos külön adathalmazt, ezért fordulhatott elő az, hogy mérésünkben a nem-projektív-lista-alapú változat eredményessége a 100%-ot is megközelítette (lásd 5. ábra). Egy ismeretlen teszt mintán ez természetesen gyengébb eredményt adott. Ezzel kapcsolatban hangsúlyozandó, hogy vizsgálatunk nem arra irányult, hogy mennyire eredményesek az algoritmusok egy ismeretlen mintán, hanem arra, hogy elméletben mennyire tudnának a tanító minta megtanulásában eredményesek lenni. Jól látszik, hogy mindhárom projektív algoritmus jelentősen elmarad az elméleti határtól, azt csak a nem projektív változat tudta megközelíteni (NonProjectiveList). A másik kiolvasható információ azt mutatja, hogy hozzávetőlegesen 7-11 ismétlés után az eredményesség szignifikánsan már nem javul, sőt az ArcEager és a ProjectiveList változatok még csökkenő tendenciát is mutatnak. Ezt az információt felhasználva későbbi tesztjeinkben egységesen 7 tanulási iterációt futtatunk.

A második kisérletben a gráf-transzformációs módszert implementáltuk. A fejlesztés itt is Java környezetben történt. A transzformációs algoritmus működéséhez 21 szabályt definiáltunk. A szabályok végretehajtása nem tetszőleges sorrendben történik, jól meghatározott végrehajtási sorrend él közöttük. A létrehozott szabályrendszer ugyan még nem teljes, de már így is elég összetett nyelvi jelenségeket le tud írni a létrehozott hierarchián keresztül. A 6. ábrán a „Most érkezett a hírrel, hogy Péter beteg” mondathoz generált függőségi gráfot láthatjuk. A létrehozott gráfon jól látható, hogy az elkészített elemző modul képes:

- az almondat határok feltárására
- a főbb mondatrészek feltárására
- a szavak lemma részének meghatározására
- a függőség különböző típusainak feltárására
- a hiányzó részek pótlására
- a függőségi fa felépítésére.

A szabályalapú megközelítés egyik jól látható előnye, hogy a megalkotott szabályok tömören, előzetes betanítás nélkül adják meg a transzformációs lépéseket.



6. ábra Minta generált függőségi gráf

5. Konkluzió

A cikkben a függőségi gráfok előállításának két fő módszerét, az átmenetalapú direkt fa konstrukciót illetve a szabály alapú gráf-transzformációs módszert mutattuk be. Mindkét módszerhez elkészítettük a program implementációt Java nyelven. A hatékonyságok teszteléséhez magyar nyelvű forrásszövegeket használtunk fel. A teszteredmények azt mutatták, hogy a nem-projektív átmenetalapú algoritmusok és a szabály alapú gráf-transzformációs módszerek is jó eredményt tudnak elérni a tanító halmaz elemzésében. Általános, nem betanított szövegeket vizsgálva a szabály alapú módszerek stabilabb működést produkáltak. A szabály alapú módszernél a szabályok manuális feltárása okozott jelentős költséget. A fejlesztés további irányaként a szabály alapú rendszerek olyan irányú kibővítését tervezzük, melyben a szabályok feltárása részben automatikusan történik tanító algoritmusok segítségével.

Köszönetnyilvánítás

A cikkben ismertetett kutató munka az EFOP-3.6.1-16-2016-00011 jelű „Fiatalodó és Megújuló Egyetem – Innovatív Tudásváros – a Miskolci Egyetem intelligens szakosodást szolgáló intézményi fejlesztése” projekt részeként – a Széchenyi 2020 keretében – az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Felhasznált irodalom

[1] Prószéki G., Számítógépes nyelvészet, 2015, Szak kiadó.
[2] Klement K. C., Frege and the Logic of Sense and Reference. 2017, Routledge. https://doi.org/10.4324/9780203953303
[3] Tesnière, L. , Eléments de Syntaxe Structurale, 1959, Klincksiek.
[4] Ágel, V. and Fischer, K. , Dependency grammar and valency theory. In: The Oxford Handbook of Linguistic Analysis, 2009 **https://doi.org/10.1093/oxfordhb/9780199544004.013.0010**
[5] Kübler, S., McDonald, R., and Nivre, J. Dependency parsing. Synthesis Lectures on Human Language Technologies 2009, 2(1):1-127. **https://doi.org/10.2200/S00169ED1V01Y200901HLT002**
[6] Nivre, J. Algorithms for deterministic incremental dependency parsing, Comput. Linguist. 2008, 34(4):513-553. **https://doi.org/10.1162/coli.07-056-R1-07-027**
[7] Nivre, J. , Hall, J. and Nilson, J., Memory-based dependency parsing. In Proceedings of CoNLL, 2007, pages 49-56.
[8] Eisner, J. M., Three new probabilistic models for dependency parsing: An exploration, Proceedings of the 16th International Conference on Computational Linguistics (COLING), 1996, pp. 340-345. **https://doi.org/10.3115/992628.992688**
[9] McDonald, R., Lerman, K. and Pereira, F., Multilingual dependency analysis with a two-stage discriminative parser, Proceedings of the 10th Conference on Computational Natural Language Learning (2006), pp. 216-220. **https://doi.org/10.3115/1596276.1596317**
[10] Kübler, S. ,McDonald,R. and Nivre, J., Dependency Parsing, 2009, Morgan & Claypool **https://doi.org/10.2200/S00169ED1V01Y200901HLT002**
[11] Gaifman, H. , Dependency systems and phrase-structure systems, Information and Control 1965, 8:304-337. **https://doi.org/10.1016/S0019-9958(65)90232-9**