Természetes nyelvi interfész menetrend- és utazástervező szolgáltatásokhoz

Kemény Boldizsár, Recski Gábor

Budapesti Műszaki és Gazdaságtudományi Egyetem
Automatizálási és Alkalmazott Informatikai Tanszék

Kivonat A `Tervezz Velem` (`TeVe`) egy a Facebook Messenger platformon nyilvánosan elérhető chatbot-alkalmazás, mely természetes nyelven írt kérdésekre reagálva a budapesti tömegközlekedéssel kapcsolatos információkat jelenít meg. Az alkalmazás elemző modulja lehetővé teszi, hogy néhány szóból álló, töredékes bemenetekben is nagy pontossággal fedezzük fel a felhasználói szándékot. Az `e-magyar` nyelvfeldolgozó pipeline intergrációjának köszönhetően a rendszer képes a bemenetben szereplő főnévi esetragokat is figyelembevenni az elemzés során, a különböző szegmentációkat rangsoroló logika pedig lehetővé teszi, hogy a gyakran előforduló, többértelmű bemenetekhez a legmegfelelőbb elemzést társítsuk. A rendszernek jelenleg 160-nál több aktív felhasználója van, nyilvánosan elérhető a https://m.me/tevepp oldalon, a teljes forráskód pedig hozzáférhető a https://github.com/kebou/tevepp oldalon.

1. Bevezetés

Ebben a cikkben a `Tervezz Velem` (`TeVe`) nevű chatbot alkalmazást mutatjuk be, melynek segítségével a felhasználó természetes nyelven kommunikálva juthat információhoz a budapesti tömegközlekedésről. A rendszer a Facebook Messenger platformon nyilvánosan elérhető, és képes a chat-programokra jellemző rövid, töredékes felhasználói kérdések értelmezésére. A bevezetést követően először bemutatjuk a `TeVe` funkcionalitását és általános felépítését (2. fejezet), ezután vesszük sorra a fontosabb modulokat. A 3. fejezet a szövegfeldolgozó komponenst mutatja be, a 4. részben pedig az egyes felhasználói kéréseket értelmező és a válaszhoz szükséges információkat lekérdező komponens részleteit ismertetjük. Mivel a beérkező kéréseknek gyakran többféle elemzése is generálható, ezért a rendszer további fontos funkciója ezek rangsorolása, melyről az 5. fejezetben írunk részletesebben. Végül a 6. fejezetben bemutatjuk, miként integráltuk az alkalmazást a Facebook Messenger platformra és hogy eddig milyen tapasztalatokkal járt a nyilvános üzem. A Messenger alkalmazás a https://m.me/tevepp linken, a rendszer forráskódja a https://github.com/kebou/tevepp oldalon érhető el.

A chatbot (konverzációs ágens) olyan program, mellyel a felhasználó természetes nyelvet használva tud interakcióba lépni, és a chatbot ezt megértve tud válaszolni, információt szolgáltatni vagy különböző feladatokat végrehajtani. A chatbotokat gyakran azonosítják az általános intelligenciát megvalósító,

tetszőleges témában párbeszédet folytató programokkal, azonban ez csak egy lehetséges alkalmazási területe a konverzációs ágenseknek [1]. A chatbotok másik nagy csoportjába azok a nyelvi interfésszel rendelkező programok tartoznak, melyek egy előre jól meghatározott témakörben képesek feladatokat ellátni. Ezeket a programokat taszk-specifikus chatbotoknak is nevezzük [2,3]. Az utóbbi időben nagy érdeklődés övezi a virtuális személyi asszisztenseket, melyek a feladat-orientált programoknál szélesebb körben képesek különböző tevékenységeket elvégezni (pl. telefonhívások indítása, internetes keresések végrehajtása, útvonal tervezése, stb.), azonban az általuk megoldható feladatok száma még erősen korlátozott, általános intelligenciát nem valósítanak meg. Ebben a cikkben egy taszk-specifikus chatbotot támogató rendszer működését írjuk le. Az ilyen alkalmazásoknak manapság legelterjedtebb felületét az azonnali üzenetküldő alkalmazások adják, az általunk bemutatott alkalmazást így a Facebook Messenger platform segítségével teszteljük.

Az effajta integráció kulcsfontosságú azért is, mert – a Business Insider riportja szerint [4] – az üzenetküldő alkalmazások jelenleg már magasabb felhasználó számmal bírnak, mint az egyes közösségi hálókhoz tartozó alkalmazások. Ezt mára már a legtöbb üzenetküldő alkalmazásokat készítő cég felismerte, és egyre többen biztosítanak az automatizált üzenetküldéssel kapcsolatban integrációs lehetőséget a fejlesztők számára. A feladat-specifikus chatbotok felhasználási területei cégek esetén leggyakrabban a customer support automatizálása, a márkahűség mélyítése, webshop kialakítása és a felhasználók informálása a vásárlásokkal vagy egyéb eseményekkel kapcsolatos információváltozásokról. A cégekkel való kapcsolattartáson kívül számos kollaborációt segítő eszköz található, melyek az üzenetváltásban harmadik félként részt véve különböző feladatok – időpont-egyeztetés, emlékeztető lista készítés, stb. – végrehajtásában segítik a beszélgetőtársakat. Továbbá léteznek olyan alkalmazások, melyek egy „másik személyként" segítenek különböző információkhoz jutni. A mi rendszerünk egy budapesti tömegközlekedést segítő chatbot-alkalmazás, mely lehetővé teszi, hogy a felhasználók rövid szöveges üzenetek segítségével végezzenek menetrend- és utazástervezést.

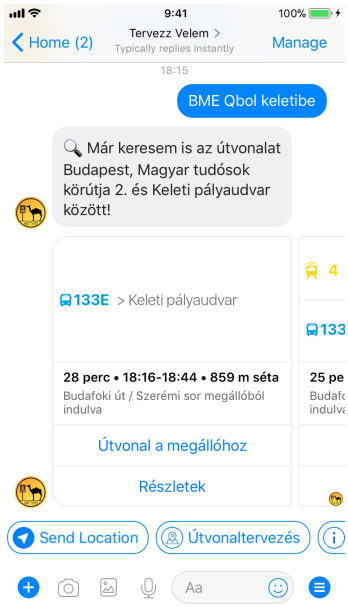
2. A Tervezz Velem rendszer

2.1. Funkcionalitás

TeVe, azaz Tervezz Velem egy chatbot, mely a Facebook által készített Messenger Platformon keresztül nyilvánosan elérhető[1]. A chatbot a budapesti tömegközlekedéssel kapcsolatban szolgáltat információkat a felhasználóknak. Segítségével megkaphatjuk a közelünkben lévő megállókból induló járatok indulási idejét, valamint a tartózkodási hely elküldése helyett meg is írhatjuk a járat és a megálló nevét, melynek menetrendjére kíváncsiak vagyunk. Szöveges üzenet alapján, vagy a helyzetválasztó segítségével megjelölt – a BKK szolgáltatása által lefedett – címek között útvonalat is tervezhetünk. Az alkalmazásban továbbá elmenthetjük

[1] https://m.me/tevepp

kedvenc helyeinket, hogy azokra később az útvonaltervezés során könnyen hivatkozhassunk. A rendszer néhány alapvető társalgási formulát is támogat, így pl. köszönésre illedelmes választ és a funkcionalitást bemutató segédüzenetet kaphatunk. Az 1. ábra egy tipikus, útvonaltervezésre irányuló párbeszédet mutat be.



1. ábra: Útvonaltervezés a chatbottal

2.2. Felépítés

A Tervezz Velem szövegfeldolgozó rendszere egy nyelvfeldolgozó, egy elemző és pontozó, valamint egy rangsoroló komponensből áll össze. A nyelvfeldolgozó modul végigfuttatja a bemeneti szöveget az `e-magyar` nyelvfeldolgozó lánc tokenizáló, lemmatizáló és morfológiai elemző moduljain (ld. részletesebben a 3. fejezetben). A nyelvfeldolgozás után egy elemzési táblázat készül, mely során az egyes mezőknek megfelelő szókapcsolatok lekérdezésre kerülnek a felhasznált API-któl, és a találatok különböző szempontok alapján pontszámokat kapnak; ennek folyamatát a 4. fejezet írja le. Ezt követően legenerálódik a bemeneti szöveg összes lehetséges felosztása, majd az egyes felosztások az elemzési tábla alapján kiértékelésre kerülnek, végül kialakításra kerül az elemzések sorrendje. A szegmentáló és rangsoroló modul működéséről az 5. fejezet szól.

3. Nyelvfeldolgozás

A szövegfeldolgozó modul a keresett információk kinyeréséhez minden lehetséges szókapcsolatot külön-külön megvizsgál, melyhez a bemeneti szöveget darabokra kell bontani A rangsoroló folyamat során kimagasló pontszámot kapnak a megfelelő ragokat tartalmazó szókapcsolatok. Egy-egy megálló- vagy utcanév esetében pl. a *-tól/-től* vagy *-hoz/hez/höz* rag egyértelmű jele annak, hogy az adott entitás egy útvonaltervezés esetében a kiindulási vagy érkezési helyet jelöli. Az efféle információk kinyeréséhez szükség van a főnévi esetragok felismerésére, az e-magyar feldolgozási lánc morfológiai elemzője ezt is lehetővé teszi [5].

A nyelvfeldolgozó modul első lépéseként a bemeneti szöveg végigfut a GATE szövegelemző rendszeren, melyben az `e-magyar` `emToken`, `emMorph`, `emTag`, `emLem` moduljai kerülnek felhasználásra. Az `emToken` modul tokenizálja a bemeneti szöveget, melynek köszönhetően később lehetségessé válik a szöveg szavankénti vizsgálata. Ezután a morfológiai elemzés következik, melyet az `emMorph` modul végez. A modul a bemenet minden szóalakjához hozzárendeli annak összes lehetséges morfológiai elemzését. Mivel minden szónak több lehetséges morfológiai elemzése is lehet, a szöveg egészét vizsgálva ki kell választani ezen elemzések közül a megfelelőt. Ezt a morfológiai egyértelműsítést az `emTag` modul végzi. Az egyes tokenek szótövezését és a szótövek szófaját az `emLem` modul határozza meg. A GATE szerver eléréséhez készítettünk egy API csomagoló klienst[2]. Ennek a kliensnek a feladata a feldolgozandó szöveg elküldése a GATE szervernek és a visszakapott válasz a továbbiakban könnyen kezelhető formátumra hozása. A GATE kliens a feldolgozási lánc `parseText` moduljából kerül meghívásra, ami az e-magyar elemzés lekérdezése után kiszűri a névelőket, írásjeleket, kötőszavakat és a kérdőnévmásokat a tokenek közül, ugyanis ezek elhagyása megkönnyíti a következő modulok feladatát.

4. Elemzés

A fő elemző modul elkészítése előtt a következő feltevésekkel éltünk:

- a szövegből kinyerendő információk az *indulási hely*, *érkezési hely* és a *járatnév*;
- ezen információk közül bármelyik kettő egyértelműen meghatározza a választ:
  - indulási és érkezési hely együttes jelenléte esetén útvonaltervezés,
  - indulási hely vagy ragot nem tartalmazó hely és egy járatnév találata esetén járatindulási információk,
  - érkezési hely és járatnév együttes előfordulásakor a kiindulási helyre irányuló kérdés;
- úgy tekintjük a szöveget, mintha abban az összes szükséges információ szerepelne – tehát egy-egy lekérdezést követően nem törekszünk további interakcióra, minden esetben előnyben részesítjük azokat az elemzéseket, melyek alapján a kérdés már megválaszolható.

[2] https://github.com/kebou/emagyar

A rendszer egy korábbi verziójában az elemzőlánc egyes moduljai csak az előttük már lefutott komponensek kimenetéhez fértek hozzá - nem elemezhették tehát azokat a tokeneket, melyeket egy magasabb precedenciájú komponens már feldolgozott. Így a rendszer nem volt képes önkorrekcióra, egy korai modul hibája a teljes elemzésre nézve végzetesnek bizonyult. Ezen hibák javítására készítettük el a feldolgozási lánc módosított változatát, melyben az összes lehetséges szókapcsolat teljes elemzését elvégezzük. Ehhez szükség van egy olyan adatstruktúrára, mely tárolja az egyes szókapcsolatokat és a hozzájuk tartozó elemzéseket is. Ennek a nyilvántartásnak létrehozása és adatokkal való feltöltése a `generateMap` modul feladata.

A nyilvántartásra használt tárolót egy kétdimenziós tömbként implementáltuk. A tömb és minden altömb elemszáma megegyezik a modul bemeneteként kapott tokenek számával, azaz egy négyzetes mátrixnak is tekinthető. Erre a kétdimenziós tömbre a továbbiakban elemzési táblázatként fogunk hivatkozni. A tömb első dimenziója a szókapcsolat kiindulási tokenjének, második dimenziója a záró tokenjének indexeit tartalmazza. Tehát például a *„Batthyány tér 1 Oktogon"* szöveg *tér 1 Oktogon* szókapcsolatának elemzését az elemzési tábla `[1][3]` indexelésével érhetjük el, a *Batthyány tér* szókapcsolatét pedig a `[0][2]`-vel. Lehetőséget teremtettünk a szókapcsolatok maximális tokenszámának korlátozására, tehát megszabhatjuk, hogy egy-egy feldolgozandó egység maximum hány szóból állhat. Ez a beállítás gyorsíthat a feldolgozáson, mivel nem vizsgáljuk meg a fölöslegesen hosszú szókapcsolatokat. Az elemzési táblázat egy kitöltésére az 1. táblázat mutat példát - a maximális tokenszám beállítása miatt a szürkével jelölt mezők üresen maradnak, ezekre nem fut le az elemző algoritmus.

	Batthyány	tér	1	Oktogon
Batthyány	(0,0)	(0,1)	(0,2)	
tér		(1,1)	(1,2)	(1,3)
1			(2,2)	(2,3)
Oktogon				(3,3)

1. táblázat. A *„Batthyány tér 1 Oktogon"* szöveg elemzési táblájának vázlata

A táblázat minden érvényes mezőjében az adott szókapcsolatot reprezentáló `MapNode` csomóponti objektum található. Ez az objektum tartalmazza a mezőhöz tartozó tokeneket, a szókapcsolathoz tartozó elemzési találatokat és a csomópont jóságát jelző pontszámot. Az egyes elemzési találatokat egy `MapElement` objektum írja le, mely tartalmazza a találat típusát, tartalmát és pontszámait. Egy csomóponti objektum értékeinek feltöltése két részben történik. Először elemzésre kerül a szövegrész, majd az összes lehetséges elemzés átfut a pontozó logikán.

4.1. Elemzés

Első lépéseként ragok keresése történik a csomóponthoz tartozó utolsó token morfológiai elemzése alapján. A megfelelő ragok jelenléte és felismerése megkönnyíti az egyes információ-egységek elhatárolását, és segítenek az adott cím vagy megálló mondaton belüli szerepének (kiindulás vagy érkezés) megállapításában is. Kiindulási helyet az *elativus (-ból, -ből)*, *delativus (-ról, -ről)* és *ablativus (-tól, -től)* ragok jelenthetnek, érkezési helyet pedig az *illativus (-ba, -be)*, *sublativus (-ra, -re)*, *allativus (-hoz, -hez, -höz)* és *terminativus (-ig)* ragok. Bizonyos esetekben előfordulhat, hogy a morfológiai elemző nem talál ragot az adott tokenen, pedig az mégis tartalmazza azt. Jó példa erre a *„Nyugati pályaudvarról"* szókapcsolat, ugyanis jelen írásmód esetén a morfológiai elemzés helyesen megadja, hogy a *pályaudvarról* szón megtalálható a *-ról* delativusi esetrag, azonban a rag ékezetének elhagyásával, a *pályaudvarrol* morfológiai elemzéséről már nem mondható el ugyanez. Az ilyen esetek helyes kezelése érdekében a rag meglétének és típusának vizsgálata reguláris kifejezésekkel is megtörténik. A rag jelenlétét és szerepét jelző információ a csomóponti objektumban kerül nyilvántartásra.

Következő lépésként lekérdezésre kerül a szókapcsolat a BKK Futár API-tól mint megálló és a Google Maps Geocoding API-tól mint cím. Sikeres lekérdezés esetén a találat elhelyezésre kerül a csomóponti objektum lehetséges elemzéseit tartalmazó tömbjébe. Egyetlen tokent tartalmazó csomópont esetén egy reguláris kifejezés ellenőrzi, hogy a token értelmezhető-e járatnévként, és ha igen, akkor ez is bekerül a lehetséges elemzések közé. Végül ha a szövegrész megegyezik a felhasználó által elmentett kedvenc helyek valamelyikével, akkor ez a hely szintén mentésre kerül az elemezések közé.

4.2. Pontozás

Az elemző komponensek lefutása után a csomóponti objektum a pontozó logikához kerül, aminek feladata a csomópont különböző szempontok szerinti pontozása a teljes bemenet lehetséges elemzéseinek későbbi rangsorolása érdekében. A szempontok súlyozásához felhasznált értékek jelenleg manuális próbálgatás eredményeként kerültek meghatározásra. Az értékelés két szinten történik: pontozásra kerülnek a csomópont szókapcsolatához tartozó elemzések, és kiértékelődik maga a csomópont is. A logika mindkét szinten moduláris felépítésű, így könnyen bővíthető, a jövőben tetszőleges újabb tulajdonságok szerint kiértékelhetőek a találatok.

Jelenleg az elemzési eredmények szintjén három, csomópontok szintjén pedig két szempont alapján történik a pontozás. Az elemzési szinten megállapításra kerül a csomóponthoz tartozó szókapcsolat és a elemzés eredményeként megkapott szöveg közti *Levenshtein-távolság*. Ez azt jelenti, hogy ha például valamelyik API a *Nyugati* sztringre a *Nyugati pályaudvar* kimenetet adja vissza, akkor e két sztring hasonlóságát vizsgáljuk. A távolság meghatározásakor a kis- és nagybetűk, illetve az ékezetes karakterek és azok ékezet nélküli párjai azonos karakternek számítanak, mivel a bemeneti szövegben ezek gyakran keveredhetnek. A

Levehnstein-távolság alapján adott hasonlósági pontszámot az (1) képlet alapján számoljuk, ahol a és b a két összehasonlítandó szövegrész, $levenshtein(a, b)$ a két szövegrész közötti szerkesztési távolság, w a pontozási szempont súlya, $|a|$ és $|b|$ pedig a két sztring karakterszámát jelöli.

$$s = w * \left(1 - \frac{levenshtein(a, b)}{max(|a|, |b|)}\right) \tag{1}$$

Szintén az egyes elemzések jóságát jelző pontszám a **relatív hossz**, mely a keresési szövegrész és az eredmény szövegrész közötti szószám összehasonlítására szolgál. Értéke a szerkesztési távolsághoz hasonlóan a (2) képlet alapján áll elő - ezúttal $|c|$ és $|d|$ a két sztring szavainak számát jelöli. A relatív hossz segítségével jutalmazhatjuk a kereséshez szószámban legközelebb álló találatot, és büntethetjük a nagyon eltérőt.

$$s = w * \left(1 - \frac{||c| - |d||}{max(|c|, |d|)}\right) \tag{2}$$

Az egyes elemzések a **találat típusa** alapján is értékelésre kerülnek. A típusra vonatkozó értékekkel megadható a találatok fontosságának sorrendje. A sorrend kialakítása történhet egyéni preferencia szerint is, azonban jelen alkalmazás esetén ezen pontozási szempont segítségével korrigálható az API-k lazán kapcsolódó találatainak kezelése is. Ugyanis a Google Maps Geocoding API-ja sokkal megengedőbb, mint a BKK Futáré, azaz kis egyezés esetén is ad vissza találatokat, ezért a megálló típusú találatok több pontot kapnak az egyes címeknél. Szintén ezzel tudjuk beállítani, hogy egy más elemzéssel egyező nevű kedvenc hely találatakor a kedvenc kerüljön kiválasztásra.

A csomóponti értékelési szempontok egyike a **rag alapú** pontozás. Ezen szempont alapján abban az esetben kap pontot a csomópont, ha tartalmaz a kiindulási vagy érkezési helyeknek megfelelő esetragot. Külön értékkel súlyozható a morfológiai elemzés révén megállapított és a reguláris kifejezések által megtalált rag, így nagyobb mértékben jutalmazhatjuk a megbízhatóbb morfológiai elemzés találatait és például a csak látszólag ragozott *Szentendre* szó nem kap túl magas pontszámot.

Az egyik legfontosabb és legérzékenyebb pontozási szempont az **abszolút hossz** alapú értékelés. Az abszolút hossz megegyezik a csomóponthoz tartozó tokenek számával. Az ezen szempont alapján adott pontszám befolyásolja, hogy milyen súllyal szerepeljenek a meghatározott hosszú szókapcsolatok a rangsorolás folyamán. Vegyük példaként a *„Fehérvári út 17"* szöveg elemzését. Abban az esetben, ha nem különböztetjük meg a csomópontokat a hozzájuk tartozó tokenek száma alapján, vagyis az abszolút hosszra mindegyik azonos pontszámot kap, akkor a rangsorolás után a legjobb eredmény ebből a három komponensből fog állni: `1) Fehérvári`, `2) út`, `3) 17`. Ennek oka egyszerűen az, hogy mindegyik szóra külön-külön is lett találat, és mivel a több szóból álló szókapcsolatok nem érnek jelentősen többet az egyszavasaknál, a három szóra kapott pontszám összege magasabb értékű, mint egy kétszavas és egy egyszavas találat pontjainak összege. Azonban ha a kétszavas szókapcsolatoknak magasabb pontszámot

adunk, már más a helyzet. A jól megválasztott pontszámmal elérhetjük, hogy három egyszavas elemzési találat pontjainak összege kevesebb legyen, mint egy kétszavas és egy egyszavas összege. Ebben az esetben a rangsorolás után a legjobb eredmény a `1) Fehérvári út` és a `2) 17` komponenseket fogja tartalmazni. Ha a három tokenből álló szókapcsolatokat jobban előnyben szeretnénk részesíteni, mint a kettőből állókat, akkor – a mintát követve – ezek abszolút hosszára olyan pontszámot kell adnunk, amely egy egyszavas és egy kétszavas szókapcsolat pontjainak összegénél magasabb. Azonban könnyen eljuthatunk egy olyan szószámhoz, ahol már nem szabad ennyire jutalmazni a hosszúságot. Már négy szó esetén is adódhatnak problémák. Tekintsük a *„Fehérvári út 17 Andrássy út"* mondatot. A Google API megengedő tulajdonsága miatt az *út 17 andrássy út* szókapcsolatra is ad vissza találatot, ami négy szóból áll. Vagyis ha a négyszavas szókapcsolatoknak magasabb pontszámot állítanánk be, mint a két és háromszavasak összege, akkor a rangsor végén a találat így alakulna: `1) Fehérvári`, `2) út 17 andrássy út`. Azonban jól látható, hogy egyáltalán nem ezt a szegmentálást szerettük volna elérni. Javíthatunk ezen a helyzeten, ha négyszavas szókapcsolatok pontszáma nem haladja meg a két és három szavasok pontszámának összegét.

Annak érdekében, hogy tartsuk magunkat ahhoz az alapfeltevésünkhöz, miszerint úgy kezeljük a szöveget, mintha abban minden információ-egység rendelkezésünkre állna, külön esetként kerül kezelésre az a szókapcsolat, mely a teljes szöveget lefedi. Az ilyen szókapcsolat kevesebb pontot kap, mint amennyit a tokenszáma alapján kapna, ugyanis előfordulhat, hogy valójában több információegységet is lefed. Visszatérve korábbi példánkhoz, bár a *„Fehérvári út 17"* szöveg önmagában is megfeleltethető egy cím típusú információ-egységnek, mivel a teljes szöveget lefedi, ezért a kisebb pontszám miatt hátrébb kerül a rangsorban és eredményül a `1) Fehérvári út` megállót és a `2) 17` járatnevet kapjuk, így a felhasználói kérdésre tudunk válaszolni: válaszul a 17-es villamos következő indulási idejét adjuk meg a Fehérvári út megállóból.

Következő lépésként a teljes felhasználói bemenetnek összes lehetséges szegmentációját rangsoroljuk a bennük szereplő csomópontok pontszámaiból kiindulva. A rangsorolást részletesen az 5. fejezet mutatja be. Az egyes csomópontokat ebben a folyamatban a legmagasabb pontszámot elért elemzésük és a csomópont saját pontértéke alapján vesszük majd figyelembe. A csomópontok pontozásakor az egyes szempontok súlyozása egyelőre „trial-and-error" módszerrel történt (ld. a 2. táblázatot), tanítóadat birtokában azonban később lehetőség nyílhat valamely *learning-to-rank* algoritmus alkalmazására is, így ezeket a paramétereket úgy tudjuk majd beállítani, hogy a lehetséges elemzések rangsorolása minél jobban megfeleljen a tényleges felhasználói szándéknak.

4.3. Elemzési példa

A 3. ábrán az elemzési táblázat feltöltésére és a pontozó logika működésére adunk példát. Látható, hogy a táblázat oszlopainak és sorainak száma megegyezik az elemzendő szöveg szószámával, illetve a főátló alatt nem tartalmaz elemeket. Minden mező első sorában a csomópontra vonatkozó információk találhatóak.

Pontozási szempont	Típus	Súlyozás
rag	morfológiai elemzés	22
	reguláris kifejezés	20
abszolút hossz	1 token	80
	2 token	220
	3 token	550
	4 token	690
	5 token	300
relatív hossz	-	10
szerkesztési távolság	-	1
elemzés típusa	járatnév	10
	megálló	10
	kedvenc hely	15
	cím	1

2. táblázat. A pontozási szempontok súlyozása

A mezőhöz tartozó indexek után a csomópont pontszáma szerepel. Ezen információkon kívül megtalálhatóak a mezőben a csomóponthoz tartozó szókapcsolat elemzései, melyeknél megálló esetén `STOP`, cím találata esetén `LOC` felirat után a megfelelő API-tól visszakapott szöveg, majd az adott elemzés két tizedesre kerekített összpontszáma szerepel. Az elemzésekkel egy cellában találhatóak még az egyes értékelési szempontok és a hozzájuk tartozó pontszám is.

A `(0,1)` indexű csomóponthoz a *Silvanus setanyrol* szöveg tartozik. Láthatjuk, hogy a szókapcsolatra két elemzési találat született, egy megálló és egy cím. Ezek közül a megállónév összpontszáma nagyobb, ezt aláhúzással jelöl a táblázat. A 21-es összpontszám komponensei: a találat típusára 10 pont, a relatív hosszra 10 pont, a szerkesztési távolságra 1 pont. A relatív hossz pontszám értéke maximális, mivel a keresendő *Silvanus setany* és a visszakapott *Silvanus sétány* szószáma egyezik. A szerkesztési távolság pontszáma szintén maximális, mivel rag találata esetén a rag nélküli szövegrész kerül keresésre és a keresett és visszakapott szövegben csupán a kis- és nagybetűk, illetve az ékezetek különböznek, melyek ilyenkor nem számítanak különböző karakternek. A csomópont a ragnak köszönhetően 20 pontot, a hozzá tartozó tokenek száma alapján abszolút hosszára pedig 220 pontot kap. Így a végső pontszáma ezen két érték és a legmagasabb elemzési találat pontszáma alapján 261 pont lesz. A többi csomópont értéke hasonlóan kerül kiszámításra. Látható, hogy a harmadik oszlop első és második sorának pontszáma negatív, mivel a *Silvanus setanyrol deak* és a *setanyrol deak* szókapcsolathoz nem tartozik elemzési találat. A táblázatban dupla aláhúzással szerepelnek azon csomópontok pontszámai, melyek a rangsorolás (ld. 5. fejezet) után a legtöbb pontot elérő szegmentációban szerepelnek.

	0 - Silvanus	1 - setanyrol	2 - deak
0	(0,0) - **95.53** - absoluteLength: 80 STOP: Silvanus sétány: **15.53** - editDistance: 0.53 - relativeLength: 5 - typeBasedScore: 10 LOC: Visegrád: **11.13** - editDistance: 0.13 - relativeLength: 10 - typeBasedScore: 1	(0,1) - **261** - absoluteLength: 220 - hasRole: 20 STOP: Silvanus sétány: **21** - editDistance: 1 - relativeLength: 10 - typeBasedScore: 10 LOC: Bp., Silvanus sétány - **8.27** - editDistance: 0.6 - relativeLength: 6.67 - typeBasedScore: 1	(0,2) - **-10000** - absoluteLength: 80 -
1	-	(1,1) - **135.43** - absoluteLength: 80 - hasRole: 20 LOC: Budapest: **11.13** - editDistance: 0.13 - relativeLength: 10 - typeBasedScore: 1	(1,2) - **-10000** - absoluteLength: 220 -
2	-	-	(2,2) - **93.6** - absoluteLength: 80 STOP: Deák Ferenc tér: **13.6** - editDistance: 0.27 - relativeLength: 3.33 - typeBasedScore: 10

3. táblázat. A *„Silvanus setanyrol deak”* szöveg elemzési táblázata

5. Rangsorolás

Az elemzéseket és pontozásokat tartalmazó táblázat elkészítése után a következő lépés a rangsorolás, mely során a teljes bemeneti szöveg összes lehetséges felosztása közül választjuk ki a megfelelőt. Elsőként legeneráljuk a bemeneti szöveg összes lehetséges szegmentálását. Ahogy az elemzési táblánál, itt is beállítható egy maximális hossz, ekkor egyetlen szegmentum sem lehet ennél hosszabb, így sok szegmentáció kizárható. A szövegnek megengedett olyan felosztása is, amely nem tartalmazza az összes tokent: a természetes nyelven megfogalmazott útvonal-tervezés például a kiindulási és érkezési helyeken felül további, az információ kinyerése szempontjából közömbös szót tartalmazhat (*Hogyan jutok el...*, stb.).

Hogy megértsük, miért van szükség minden lehetséges felosztásra, tekintsük az *„Albertfalva utca 17”* szöveget. Ennek a bemenetnek kétféle olvasata is lehetséges: ha egyetlen szegmentumként kezeljük, egy pontos címet értünk alatta, egy másik értelmezés szerint a felhasználó a *17-es* villamos indulására kíváncsi az *Albertfalva utca* megállóból. Ahhoz, hogy mindkét értelmezés később pontozásra kerüljön, elő kell állítani az összes értelmezési lehetőséget. A két lehetséges ér-

telmezés szürke színnel van jelölve az összes lehetséges felosztást tartalmazó 4. táblázatban.

Index párok	Infromáció-egységek
(0,0)	[Albertfalva]
(0,1)	[Albertfalva utca]
(0,2)	[Albertfalva utca 17]
(0,0), (1,1)	[Albertfalva], [utca]
(1,1)	[utca]
(0,0), (1,2)	[Albertfalva], [utca 17]
(1,2)	[utca 17]
(0,0), (2,2)	[Albertfalva], [17]
(0,1), (2,2)	[Albertfalva utca], [17]
(0,0), (1,1), (2,2)	[Albertfalva], [utca], [17]
(1,1), (2,2)	[utca], [17]
(2,2)	[17]

4. táblázat. Az *„Albertfalva utca 17”* szöveg lehetséges particionálásai

A rangsoroló logika minden egyes szegmentációhoz az elemzési tábla alapján egy számot rendel. Egy szegmentációhoz úgy rendelünk pontszámot, hogy az abban szereplő indexpároknak az elemzési táblában megfelelő csomópontok pontszámait összeadjuk. Tehát az `[Albertfalva utca], [17]` sor értéke pl. az elemzési tábla `[0][1]` és `[2][2]` mezőiben található pontszámok összege.

6. A chatbot alkalmazás

Befejezésül bemutatjuk, hogyan integráltuk az eddigiekben leírt architektúrát egy a Facebook Messenger platformon működő chatbot-alkalmazásba és milyen tapasztalatokkal gazdagodtunk a nyilvános tesztelés során.

Az alkalmazáslogika Node.js – szerver oldali JavaScript – nyelven íródott és egy virtuális gépen fut. Amikor a felhasználó üzenetet küld a Messenger alkalmazáson keresztül, a Facebook továbbítja azt az általunk megadott webhook URL irányba. Az URL-re érkező kéréseket az alkalmazáslogika üzenettípusok szerint dolgozza fel. A Messenger lehetőséget kínál a szöveges üzenet mellett egyéb üzenettípusok küldésére is: a felhasználó küldhet képeket, videókat, különböző csatolmányokat, illetve választhat a chatbot által kínált gyors-válaszok közül, és használhatja az egyes üzenetsablonokon elhelyezett gombokat is.

Az alkalmazás magját a BKK Futár API-ja képezi, mely az egyes megálló és járatnevek validációjáért, az indulási információkért és az utazástervezésért egyaránt felel. Használatával számos további információt elérhetünk a budapesti tömegközlekedéssel kapcsolatban: forgalmi változások, járatok aktuális pozíciója, tervezett és tényleges indulások. A megállón kívüli egyéb helyek keresésére a

Google Maps Geocoding API-ját használjuk, ennek segítségével a szövegesen megadott címekből egységesen strukturált, koordinátákkal ellátott objektumokat kaphatunk.

Az adatok tárolása relációs adatbázis helyett MongoDB NoSQL dokumentum alapú adatbázisban történik, mely jelentősen megkönnyíti a különféle forrásból származó adatok konzisztens mentését. A bemeneti szöveg feldolgozása során jelentős szerepet játszik a program mellett futó GATE szerver és a hozzá kapcsolódó e-magyar feldolgozási lánc moduljai is.

6.1. Az elemző modul integrációja

A chatbot szövegelemző komponensének elkészítése során az átstrukturálhatóság és a későbbi bővíthetőség érdekében egy könnyű, moduláris feldolgozási lánc megvalósítását tartottuk szem előtt. A feldolgozási sor egyes komponensei azonos interfészt valósítanak meg: minden modul meghíváskor két paramétert kap, az egyik egy, a feldolgozási láncon végigmenő, közös kontextus objektum, amelybe minden modul tetszőlegesen helyezhet új értékeket és amelyből bármely értéket felhasználhat. A másik paraméter egy, a következő modul hívására szolgáló függvény. Amennyiben ez a függvény nem kerül meghívásra, illetve nincsenek további modulok, a feldolgozási sor véget ér.

A chatbot alkalmazásához beérkező üzenet először végigfut a `wit.ai` szolgáltatáson, mely az alapvető felhasználói szándék felismerésére szolgál. Ennek az elemezésnek köszönhetően kerülnek feldolgozásra a köszönések, funkció hívások és segítségkérések. Ha ilyen felhasználói szándékot nem azonosítunk, működésbe lép a korábbi fejezetekben ismertetett logika. Először a nyelvfeldolgozó modul az `e-magyar` segítségével feldolgozza a szöveget, majd feltöltésre kerül az elemzési táblázat. Végül legenerálódnak a lehetséges felosztások és megtörténik a rangsorolás.

A kialakult rangsor alapján a következő modul meghatározza a kiindulási helyek, érkezési helyek és járatnevek lehetséges értékeit és elhelyezi azokat a kontextusban. Ezen lépés után olyan modulok futnak le, melyek ha megtalálnak minden, a működésük számára szükséges értéket a kontextusban, akkor feladatuknak megfelelő üzenetet küldenek a felhasználónak. Amennyiben egyik komponens sem talál elegendő információt, azaz nem sikerült értelmezni a felhasználó üzenetét, egy erről szóló üzenet kerül kiküldésre.

6.2. A chatbot használat közben

A chatbot nyilvános tesztelése 2017. októberében indult, az első 6 hét után több, mint 170 aktív felhasználóval rendelkezik és ez idő alatt 16000 beérkező üzenetet dolgozott fel. Átlagosan napi két-három felhasználó tervez utazást vagy keres menetrendi információkat segítségével. A Tervezz Velem chatbot az elemző modul egy korábbi – a rangsoroló logikát még nem tartalmazó – verziójával kezdte meg a nyilvános működést. A cikkben bemutatott új modul bekötése után jelentős javulást várunk az egyes információ-típusok felismerésével kapcsolatban. A

lehetséges továbbfejlesztések közül az utazástervezés időpontjának beállíthatósága szerepel még rövidtávú terveink között, a felhasználói visszajelzések alapján ugyanis erre mutatkozik a legnagyobb igény.

7. Köszönetnyilvánítás

Köszönjük a SZTAKI Nyelvtechnológiai Kutatócsoportjának, hogy biztosították a nyilvános teszteléshez szükséges infrastruktúrát, valamint két névtelen bírálónknak a hasznos észrevételeket.

Hivatkozások

1. Carpenter, R.: Cleverbot. http://www.cleverbot.com (2017) Letöltés ideje: 2017.12.05.
2. Weizenbaum, J.: ELIZA – A Computer Program For the Study of Natural Language Communication Between Man and Machine. Communications of the ACM **9**(1) (1966) 36–45
3. Colby, K.M., Weber, S., Hilf, F.D.: Artificial paranoia. Artificial Intelligence **2**(1) (1971) 1–25
4. BI Intelligence: Messaging apps are now bigger than social networks. http://www.businessinsider.com/the-messaging-app-report-2015-11 (2017) Letöltés ideje: 2017.11.13.
5. Váradi, T., Simon, E., Sass, B., Gerőcs, M., Mittelholcz, I., Novák, A., Indig, B., Prószéky, G., Farkas, R., Vincze, V.: Az e-magyar digitális nyelvfeldolgozó rendszer. In: XIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2017), Szeged, Szegedi Tudományegyetem (2017) 49–60