Jedno rešenje dekodera za automatsko prepoznavanje govora na velikim rečnicima

Nikša M. Jakovljević, Dragiša M. Mišković, Marko B. Janev i Darko J. Pekar

***Sadržaj* — U ovom radu je predstavljena struktura dekodera namenjenog automatskom prepoznavanju govora na velikim rečnicima. Dekoder je implementiran u programskom jeziku C++, a celokupan kod je dokumentovan i javno dostupan. Osnovne tehnike koje se koriste u procesu prepoznavanja na velikim rečnicima su realizovane. Zbog svoje modularne strukture, modifikovanje i proširenje koda je relativno jednostavno. Pošto su podržani standardni formati koji se koriste u prepoznavanju govora, moguće je njegovo korišćenje u kombinaciji sa drugim javno dostupnim alatima.**

***Ključne reči* — prepoznavanje govora, dekoder, softver**

I. UVOD

ZA potrebe automatskog prepoznavanja govora, u poslednjih dvadesetak godina, razvijeno je nekoliko javno dostupnih alata kao što su HTK [1], Sphinx-4 [2], Julius [3] i RWTH ASR [4]. Najpotpuniju dokumentaciju poseduje HTK, što u velikoj meri olakšava njegovo korišćenje, ali je pisan u programskom jeziku C što za posledicu ima relativno složenu strukturu koda, čime je delimično otežano proširivanje i modifikovanje koda. Sphinx-4 i RWTH ASR su realizovani korišćenjem objektno orjentisanih programskih jezika (JAVA i C++ respektivno) čime je pojednostavljeno proširivanje i modifikovanje koda, ali je prateća dokumentacija relativno skromna. Za razliku od prethodno pomenutih alata koje sačinjavaju kako alati za obuku modela tako i alati za dekodovanje Julius je alat koji omogućava isključivo dekodovanje. Podržani su standardni formati, tako da se može koristiti sa drugim standardnim alatima kao što su HTK, CMU-Cam SLM i drugi.

Zbog prethodno pomenutih ograničenja pojedinih alata, kao i nemogućnosti njihovog korišćenja u komercijalne svrhe, na Fakultetu tehničkih nauka u kooperaciji sa firmom Alfanum doo. razvijen je dekoder za automatsko prepoznavanje govora na velikim rečnicima (rečnici koji broje nekoliko desetina hiljada reči). Ovaj dekoder je, kao i većina savremenih sistema za prepoznavanje govora, namenjen statistički modelovanom govoru.

Jedna od prednosti statističkog pristupa modelovanju govora je mogućnost raščlanjivanja složenog modela, primenom Bajesovog pravila, na jednostavnije podmodele, od kojih su najvažnija tri: akustički model (*acoustic model*), model jezika (*language model*) i model načina izgovora (*pronunciation model*). Akustički model daje vezu između akustičkih opservacija govornog signala i modela alofona (koji se aproksimiraju modelima fonema zavisnim od konteksta) i za ove potrebe se koriste skriveni Markovljevi modeli u kombinaciji sa mešavinama Gausovih raspodela. Jezički model sa druge strane definiše moguće sekvence reči, odnosno njihove verovatnoće pojavljivanja i obično se modeluje pomoću *n*-grama (standardno bigrama i trigrama). Model načina izgovora definiše preslikavanje reči u sekvence akustičkih modela. U slučaju sistema za prepoznavanje na velikim rečnicima prethodno nabrojani podmodeli su velike složenosti, te je za prepoznavanje u realnom vremenu neophodno obezbediti efikasnu strukturu koja će smanjiti prostor pretrage. Realizacija ovih struktura u dekoderu koji je predmet ovog rada je data u odeljku II.

Brzina dekodovanja zavisi i od načina na koji se vrši dekodovanje. Većina savremenih sistema se zasniva na algoritmima koji se mogu podeliti u dve velike grupe: „prvo u dubinu“ (depth-first) i „prvo u širinu“ (breadth-first). Kod algoritama iz klase „prvo u dubinu“ vrši se ekspandovanje u tom trenutku najverovatnije putanje, a najčešće korišćen algoritam iz ove klase u prepoznavanju govora je *stack* algoritam. Alogoritmi klase „prvo u širinu“ istovremeno proširuju nekoliko putanja. Najzastupljeniji algoritam iz ove klase je Viterbijev algoritam. U slučaju prepoznavanja govora na velikim rečnicima, neophodno je broj potencijalnih putanja držati konstantnim, odnosno vršiti tzv. *pruning*.

Dekoder predstavljen u ovom radu se zasniva na modifikovanom Viterbijevom algoritmu, tzv. *token-passing* algoritmu [5]. Detalji vezani za specifičnost ove implementacije dati su u odeljku III.

II. ORGANIZACIJA PROSTORA ZA PRETRAŽIVANJE

U slučaju prepoznavanja govora na malim rečnicima, svaka reč je predstavljena kao linearna sekvenca fonema nezavisno od drugih reči (Sl. 1). Ovakva organizacija, koja se obično naziva linearni leksikon, primenjena u sistemima

Ovaj rad je realizovan u okviru projekta TR-11001 Ministarstva za nauku i tehnološki razvoj republike Srbije.

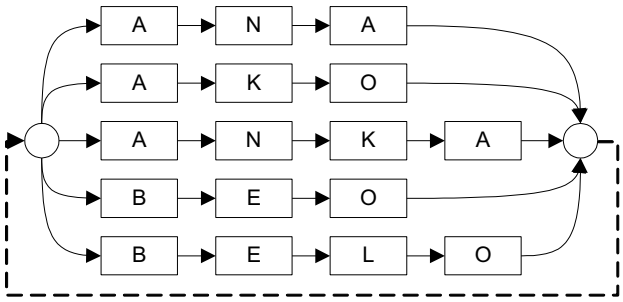
Nikša M. Jakovljević (autor za kontakte), Fakultet tehničkih nauka Novi Sad, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (telefon: 381-21-485-2521, e-mail: jakovnik@uns.ac.rs)

Dragiša M. Mišković, Fakultet tehničkih nauka Novi Sad, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (telefon: 381-21-485-2521, e-mail: dragisa.miskovic@alfanum.co.rs

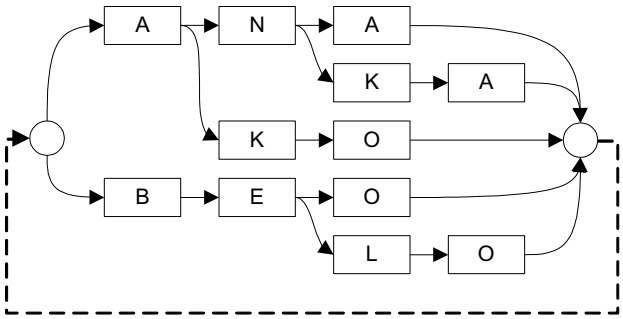
Marko. B. Janev, Fakultet tehničkih nauka Novi Sad, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (telefon: 381-21-485-2521, e-mail: marko.janev@alfanum.co.rs)

Darko J. Pekar, Alfanum doo. Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (telefon: 381-21-475-0080, e-mail: darko.pekar@alfanum.co.rs)

za prepoznavanje govora na velikom broju reči je nepraktična pošto nepotrebno povećava prostor pretrage, jer u slučaju velikog broja reči mnoge podsekvence fonema su zajedničke. Ovu činjenicu maksimalno koriste sistemi zasnovani na „pretvaračima sa konačnim brojem stanja“ (*finite state transducers*) [6]. Deljenje proizvoljnih podsekvenci fonema u reči značajno komplikuje algoritam pretrage (identifikaciju reči kroz koju je putanja prošla), stoga se pristupa kompromisnom rešenju u kome reči dele samo iste segmente koji se nalaze na početku reči (Sl. 2). Ovakva organizacija se naziva leksičko ili fonetsko prefiksno stablo (*lexical or phonetic prefix tree*) [7].



Sl. 1. Mreža reči – Ilustracija organizacije prostora za pretraživanje u slučaju prepoznavanja na malim rečnicima



Sl. 2. Leksičko stablo – Reči sa Sl 1. organizovane u leksičko stablo

Prednost korišćenja leksičkog stabla je u redukciji prostora pretrage (npr. za skup od 12k reči engleskog jezika, broj čvorova koji reprezentuju foneme je smanjen sa 101k na 43k [8]). Sa druge strane, za razliku od linearnog leksikona, informaciju o reči kroz koju putanja prolazi moguće je dobiti tek kada putanja uđe u poslednji fonem reči, što za posledicu ima povećanje broja hipoteza zbog nemogućnosti pravovremene primene modela jezika. Ovaj nedostatak se prevazilazi primenom tzv. faktorisanog modela jezika [9].

U slučaju kada se koristi model jezika koji je složeniji od unigrama, tada verovatnoća pojave reči zavisi i od reči koje su joj prethodile, tako da se verovatnoće modela jezika ne mogu direktno povezati sa odgovarajućim čvorovima u leksičkom stablu. U literaturi su predložena dva rešenja. Prvo rešenje podrazumeva ekspanziju stabla, odnosno da se na kraj svake reči doda kopija stabla koja sadrži samo one reči koje mogu da slede nakon te reči odnosno tih prethodnih reči. Na prvi pogled, ovakva ekspanzija značajno povećava prostor za pretragu (neka je N ukupan broj reči, D ukupan broj čvorova u leksičkom stablu, tada ekspanzija u slučaju bigrama daje $D(N+1)$ čvor, odnosno trigrama $D(1+N+N^2)$ čvor), ali pošto nisu moguće sve kombinacije reči ukupan broj čvorova se uveća svega nekoliko stotina puta. Drugo rešenje podrazumeva da leksičko stablo definiše samo moguće putanje, a faktorisane verovatnoće modela jezika se određuju na osnovu podataka u kom čvoru leksičkog stabla se aktivna hipoteza (potencijalna putanja) nalazi i kroz koje je reči prethodno prošla. U realizaciji dekodera koji je predstavljen u ovom radu, opredelili smo se za drugo rešenje.

Struktura leksičkog stabla je realizovana kroz klasu *CLexicalTree.* Osnovni podatak član ove klase je *m_PhonemeNodes,* koji sadrži sve čvorove leksičkog stabla. Informacije o strukturi stabla se čuvaju lokalno u čvorovima. Ovi čvorovi su realizovani kao klasa *ClexicalTreeNode*. Svaki čvor sadrži informacije: *i*) kom fonemu odgovara čvor (*m_phonemeId*), *ii*) kojoj reči odnosno kojim rečima pripada čvor (*m_wordIds*) *iii*) koji je sledeći čvor odnosno koji su sledeći čvorovi (*m_successiveNodes*) *iv*) koji su odgovarajući akustički modeli (*m_stateSeqIds*).

Pored samih čvorova, klasa *ClexicalTree* čuva informacije o tome koji su čvorovi na početku reči (*m_startNodes*), kraju reči (*m_endNodes*) kao i čvorovima tišine. Ovi čvorovi su bitni pošto imaju poseban tretman prilikom dekodovanja. Postoje 3 čvora koji predstavljaju tišinu, a to su tišina na početku rečenice (*m_startSilNode*), tišina na kraju rečenice (*m_endSilNode*) i tišina koja predstavljva pauzu između reči (*m_silNode*). Za razliku od čvora tišine na početku i na kraju rečenice, tišina koja se nalazi između reči nije obuhvaćena modelom jezika, stoga se treba ignorisati pri izračunavanju verovatnoća modela jezika. Čvor koji predstavlja tišinu na kraju reči nema naredne čvorove, a čvor koji predstavlja tišinu na početku reči nema pretodne čvorove.

Leksičko stablo se generiše na osnovu rečnika izgovora (*pronunciation dictionary*) koji se učitava u strukturu koju definiše klasa *CWordDictionary*. Ova klasa sadrži reči u njihovom pisanom i „izgovornom“ obliku. Pod pisanim oblikom reči podrazumeva se oblik reči koji se pojavljuje u tekstovima, dok se pod „izgovornim“ oblikom reči podrazumeva sekvenca odgovarajućih fonema. Srpski jezik ima fonetsko pismo, tako da se u ovom slučaju pisani i „izgovorni“ oblik reči u velikoj meri poklapaju, razlike su posledica uvođenja novih glasova radi boljeg akustičkog modelovanja (npr: razlikovanje naglašenih i nenaglašenih vokala, uvođenje neutralnog vokala šva i sl.). Treba napomenuti da se u ovom tekstu pod terminom fonem podrazumeva klasa nezavisna od konteksta koju treba modelovati.

Jedna reč može da ima jedan ili više „izgovornih“ oblika. Na osnovu „izgovornog“ oblika reči formira se leksičko stablo, tako da jedna reč može da obrazuje nekoliko putanja u stablu. Zbog svojih prethodno pomenutih posebnosti, čvorovi za tišinu se dodaju naknadno u leksičko stablo.

Verovatnoće pojavljivanja reči koje su navedene u rečniku izgovora se čuvaju u klasi *CLanguageModel*.

Trenutno je podržano modelovanje unigrama, bigrama i trigrama, ali proširivanjem funkcije za učitavanje modela jezika i dodavanjem odgovarajućih funkcija za preračunavanje verovatnoća reči postojeći kod se može jednostavno prilagoditi i za korišćenje *n*-grama složenijih od trigrama. N-gram se zadaje u standardnom ARPA formatu u kome se pored uslovnih verovatnoća n-grama, navedene i uslovne verovatnoće i *backoff* koeficijenati jednostavnijih modela. Način izračunavanja uslovnih verovatnoća koje nisu eksplicitno zadate na osnovu raspoloživih verovatnoća jednostavnijih modela i *backoff* koeficijenata je preuzet iz [10]. Prilikom učitavanja n-grama uzimaju se u obzir samo one reči koje su navedene u rečniku izgovora. Da ne bi došlo do ozbiljnijeg narušavanja statistika jezika, pri formiranju n-grama treba voditi računa da se prosledi spisak reči koje će se prepoznavati (reči koje su navedene u rečniku izgovora). U cilju ubrzanja rada dekodera, svakoj reči je dodeljen jedinstveni identifikator, s kojim se interno operiše u okviru programa, tako da se pri izračunavanju uslovnih verovatnoća modela jezika klasi *CLanguageModel* prosleđuju idenitifikatori reči umesto nizova karaktera. Prebacivanje u nizove karaktera se vrši u finalnom koraku dekodovanja, pri generisanju izlaza za korisnika.

III. Organizacija dekodera

Kao što je prethodno navedeno, ovde predstavljen dekoder bazira se na *token-passing* algoritmu (varijanti Viterbijevog algoritma gde se informacije o putanji, prethodni čvor u trelisu i akumulisana verodostojnost, ne pamte na nivou čvora trelisa, već na nivou hipoteze o putanji koja je sadržana u okviru tokena.

Token je realizovan preko klase *CToken*, koja sadrži sledeće podatke: akumuliranu verodostojnost (*m_score*), identifikator prethodnog tokena (*m_previous*), trenutak (redni broj opservacije) na koji se odnosi akumulirana verodostojnost (*m_time*), identifikator pripadajuće reči (*m_wordId*), identifikator u kom čvoru leksičkog stabla se nalazi (*m_current_node*), identifikator narednog čvora (*m_next_node*), identifikator odgovarajućeg akustičkog modela i akustičkog stanja u okviru njega (*m_model*, *m_postition*), identifikator putanje (*m_pathId*), kao i identifikator validnosti tokena (*m_bActive*)

Za razliku od originalnog Viterbijevog algoritma gde se pamti informacija o prethodnom stanju, *token-passing* algoritam omogućava da to bude čvor leksičkog stabla odnosno reč. Pored toga pošto je u svakom trenutku poznata trajektorija svih hipoteza mnogo je jednostavnije realizovati prepoznavač sa N-najboljih (*N-best*) hipoteza u odnosu na slučaj da se primenjuje originalni Viterbijev algoritam. Informacije o trenutnom čvoru i narednom čvoru leksičkog stabla kao i pozicije u akustičkom modelu su neophodne za propagaciju tokena kroz leksičko stablo (ako se nalazi u krajnjem stanju nekog modela, token treba da pređe u naredni čvor) kao i pri određivanju najverovatnije sekvence reči (nakon što se obrade sve opservacije kao potencijalne hipoteze se uzimaju one koje se nalaze u krajnjim stanjima krajnjih čvorova u stablu). Da ne bi došlo do ekspanzije broja putanja neophodno je putanje koje završe u istom stanju istog čvora i pri tome imaju istu istoriju (sekvenca reči koja im prethodi) spojiti u jednu (ostaviti onu koja ima najveću vrednost verodostojnosti). Informaciju o istoriji je moguće dobiti na osnovu informacija o prethodnim tokenima (*m_previous*), ali pošto je vreme kritičan faktor definisana je nova promenljiva *m_pathId* koja sadrži traženu informaciju.

Propagacija tokena i samo dekodovanje je realizovano u okviru klase *CFirstPassDecoder*. Dekodovanje pored informacije o tekućim tokenima (*m_vTokenList*) i prethodnim tokenima (*m_tokenHistory*) objedinjuje akustičke modele (*m_pAcousticModels*) sa lingvističkim modelima (*m_pLexTree, m_pLanguageModel*).

Akustički model je realizovan kroz klasu *CAcousticModels*. Trenutno su podržani samo modeli zasnovani na skrivenim Markovljevim modelima i Gausovim raspodelama (kako sa dijagonalnim tako i sa punim kovarijansnim matricama). Da bi se obezbedila univerzalnost ovog dekodera, ulazni format akustičkih modela je HTK tekstualni format [1]. Da bi se ubrzalo izračunavanje akustičkih verodostojnosti i obezbedilo optimalno korišćenje memorijskih resursa, ulazni skup modela se redukuje samo na potrebni skup koji je definisan skupom reči navedenim u rečniku izgovora. Komunikacija između klase *CFirstPassDecoder* i njoj pridruženog skupa akustičkih modela vrši se preko vektora *m_vAcousticProb* koji sadrži vrednosti akustičkih verodostojnosti za tekući frejm i funkcije *GetTransitionLogProb* koja vraća verovatnoću ostanka u tekućem stanju i prelaska u naredno stanje (pretpostavlja se da je struktura skrivenog Markovljevog modela sekvencijalna).

Prepoznavanje na velikim rečnicima je nemoguće realizovati u realnom vremenu bez odbacivanja manje verovatnih hipoteza tzv. *pruning*-a. Obezbeđena su dva načina *pruning*-a: na osnovu vrednosti akumulirane verodostojnosti i na osnovu maksimalnog broja aktivnih hipoteza. *Pruning* na osnovu vrednosti akumulirane verodostojnosti podrazumeva odbacivanje svih putanja čija je akumulirana verodostojnost manja od vrednosti praga koji se dobija tako što se od akumulirane vrednosti najverovatnije putanje oduzme fiksna unapred zadata vrednost (*pruning_value*). Kritična mesta su mesta na kojima dolazi do grananja putanja, čvorovi leksičkog stabla, a posebno krajnji čvorovi, stoga se za njih definišu posebne vrednosti praga. U ovom kodu je to rešeno uvođenjem penala za koje se umanjuje akumulirana vrednost prilikom prelaska tokena u novu reč, odnosno novi čvor (*word_enter_penalty* i *phone_enter_penalty*). Poseban tretman imaju čvorovi tišine, te su za njih uvedeni penali: *word_enter_penalty_sil* i *phone_enter_penalty_sil*. Fiksiranje maksimalnog broja aktivnih hipoteza podrazumeva ostavljanje samo zadatog broja najverovatnijih hipoteza, što zahteva sortiranje tokena, što čini ovaj način zahtevnijim od prethodnog. Da bi se ovo sortiranje realizovalo brzo, promenljiva *m_tokenPointers* sadrži pointere na tokene iz liste (*m_vTokenList*), koji se sortiraju umesto originalne liste tokena. Treba napomenuti da pogrešno izabrani, prethodno opisani, parametri mogu

dovesti do grešaka pri dekodovanju.

Dekodovanju prethodi proces inicijalizacije, koja podrazumeva: zauzimanje potrebnih memorijskih resursa, povezivanje modela i postavljanje tokena u čvor odnosno stanje koje odgovara tišini na početku reči. Potom se za svaku opservaciju iz sekvence opservacija primenjuju sledeći koraci: *i*) propagacija tokena, *ii*) sažimanje sličnih tokena *iii*) evaluacija akustičkih verodostojnosti i *iv*) *pruning*.

Način propagacije tokena zavisi od toga u kom stanju akustičkog modela se token nalazi. Ako se ne nalazi u krajnjem stanju nekog čvora tada se propagacija svodi na kloniranje tokena i korekciju verodostojnosti (*m_score*) i trenutka (*m_time*) ova dva tokena odgovarajućim vrednostima verovatnoće ostanka i prelaza odnosno tekućeg vremena. Token kome je vrednost verodostojnosti korigovana za vrednost prelaza uvećava vrednost *m_postition* za jedan. Sa druge strane, ako se token nalazi u krajnjem stanju nekog čvora tada je neophodno višestruko klonirane tokena (broj klonova jednak je broj narednih čvorova). Za token koji ostaje u tom krajnjem stanju vrednost verodostojnosti (*m_score*) se uvećava za verovatnoću ostanka, dok se za ostale tokene uvećava za odgovarajuće vrednosti logaritma faktorisanih verovatnoća modela jezika. Za tokene koji ulaze u nove čvorove vrši se postavljanje i odgovarajućih vrednosti za parametre *m_current_node*, *m_next_node*, *m_model* i *m_postition*. Sve ove informacije se dobijaju pomoću funkcije *GetPropagationData* klase *CLexicalTree*. Naravno za sve tokene se postavlja i odgovarajuća vrednost za *m_time*. U slučaju da token izlazi iz krajnjeg čvora reči, tada je potrebno tokenu koji izlazi postaviti odgovarajući identifikator reči *m_wordId*, identifikator putanje *m_pathId*, sam token smestiti u *m_tokenHistory*. Primetiti da sve dok token ne stigne do čvora koji predstavlja kraj reči identifikator reči upućuje na prethodnu reč.

Tokene koji se nalaze u istom stanju (*m_postition*) istog čvora (*m_current_node*, *m_next_node*) i imaju istu predsekvencu reči potrebno je sažeti u jedan omogućujući propagaciju samo onog tokena sa najvećom verodostojnosti, što je realizovano pomoću *CancelSimilarPahts* funkcije. Evaluacija akustičkih modela podrazumeva detektovanje aktivnih stanja (na osnovu liste aktivnih tokena) označavanje njihovih pozicija u vektoru *m_vAcousticProb* i prosleđivanje tog vektora funkciji *ScoreStates* klase *CAcousticModels*. Na osnovu izračunatih vrednosti akustičke verodostojnosti vrši se korekcija verodostojnosti aktivnih tokena. Proces *pruninga* je opisan u prethodnom odeljku.

Nakon što se obrade sve opservacije, od tokena koji se nalaze u krajnjem stanju krajnje tišine bira se onaj sa najvećom verodostojnošću (ili nekoliko njih u slučaju *N-best*), postavlja se odgovarajući identifikator reči i na osnovu *m_previous* određuje prethodni token. Propagacija unazad kroz tokene koji se nalaze u *m_tokenHistory* se vrši dok se ne stigne do tokena koji se nalazio u startnoj tišini. Svaki od tokena sadrži identifikator reči (*m_wordId*), trenutak kraja reči (*m_time*) i do tog trenutka akumulirana verodostoljnost (*m_score*) na osnovu čega je moguće rekonstruisati sekvencu reči, granice između reči kao i verodostojnosti pojedinih reči.

IV. Zaključak

U ovom radu je prikazana inicijalna verzija dekodera namenjenog automatskom prepoznavanju govora na velikim rečnicima, tako da su pojedini detalji podložni promeni, ali sama organizacija klasa je fiksna. U sistemu su trenutno implementirani osnovni algoritmi za dekodovanje, dok implementacija naprednih tehnika koje ubrzavaju rad i tačnost dekodera tek predstoji, te su performanse ovog sistema nešto lošije od performansi postojećih sistema (npr. Juliusa). Za razliku od većine postojećih sistema ovaj sistem podržava rad sa punim kovarijansnim matricama.

Literatura

[1] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P.Woodland, *The HTK Book*. Cambridge University Engineering Department, 2006.

[2] W. Walker, P. Lamere, P. Kwok, R. S. Bhiksha Raj, E. Gouvea, P. Wolf, and J. Woelfel, "Sphinx-4: A fexible open source framework for speech recognition," Sun Microsystems, Inc, Tech. Rep. SMLI TR-2004-139, Nov. 2004.

[3] A. Lee, T. Kawahara, and K. Shikano, "Julius – an open source real-time large vocabulary recognition engine," in *Proc. European Conf. on Speech Communication and Technology*, Aalborg, Denmark, Sep. 2001, pp. 1691–1694.

[4] D. Rybach, C. Gollan, G. Heigold, B. Hoffmeister, J. Lööf, R. Schlüter and H. Ney, "The RWTH Aachen University Open Source Speech Recognition System," in *Proc. Interspeech 2009,* Brighton, UK, Sep. 2009, pp. 2111–2114.

[5] S. J. Young, N. H. Russell, J. H. S. Thornton, "Token Passing: a Simple Conceptual Model for Connected Speech Recognition," Cambridge University Engineering Department, Cambridge, UK, Tech. Rep. CUED/F-INFENG/TR-38, July 1989.

[6] M. Mohria, F. Pereirab and M. Rileya, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language* Vol. 16, No 1, pp. 69-88, Jan. 2002.

[7] X. L. Aubert, "An overview of decoding techniques for large vocabulary continuous speech recognition," *Computer Speech & Language* Vol. 16, No 1, pp. 99-114, Jan. 2002.

[8] Ney, H. and X. Aubert, "A Word Graph Algorithm for Large Vocabulary," in *Proc. Int. Conf. on Spoken Language Processing*, Yokohama, Japan, 1994, pp. 1355-1358.

[9] F. Alleva, X. Huang, and M. Y. Hwang, "Improvements on the Pronunciation Prefix Tree Search Organization," in *Proc. Int. Conf. on Acoustics, Speech and Signal Processing*, Atlanta, Georgia 1996, pp. 133-136.

[10] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to speech recognition, computational linguistic and natural language processing.* Draft 2007, ch. 4.

Abstract

This paper describes a large vocabulary continuous speech recognition decoder. The decoder is an open source project developed in C++. Standard schemes in speech recognition are implemented. The decoder has modular structure, therefore the modifications and expansions of the code are relatively simple. It supports standard speech recognition formats, so it can be used in combination with other open source tools for speech recognition.

ONE SOLUTION FOR LARGE VOCABULARY AUTOMATIC SPEECH DECODER

Nikša M. Jakovljević, Dragiša M. Mišković,
Marko B. Janev, Darko J. Pekar