UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA INFORMATIKY

MULTIMEDIÁLNA ČÍTANKA

DTW algoritmus na transkripciu súvisle prečítaného textu s využitím transkripcie po slovách prečítaného toho istého textu

(diplomová práca)

PETER MAŇKA

štúdijný odbor: Informatika (9.2.1)

Školiteľ: RNDr. Marek Nagy

Bratislava, 2007

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím uvedenej literatúry.

V Bratislave, 27. Apríla 2007 .....

Peter Maňka

Poďakovanie

Chcem poďakovať hlavne svojmu diplomovému vedúcemu Marekovi Nagyovi, za jeho dobré nápady, pomoc pri vytváraní tejto práce a trpezlivé odpovedanie na moje otázky. Poďakovanie patrí aj Jánovi Šefránkovi z FMFI UK za jeho svedomitý prístup k výučbe diplomového seminára a Milanovi Ruskovi z UTRR SAV za konzultácie v počiatočných fázach mojej práce a poskytnutie literatúry. Ďakujem svojím spoluštudentom a priateľom za ich nápady a poznámky, ktorými ma často inšpirovali. Ďakujem svojmu spolubývajúcemu Johnymu za toleranciu a snahu vytvoriť dobré pracovné prostredie na premýšľanie a písanie. Poďakovanie patrí aj mojim rodičom, vďaka ktorým som mohol študovať.

Abstrakt

Podarilo sa mi úspešne implementovať nástroj na riešenie problému automatickej transkripcie súvisle prečítaného textu s využitím znalosti existujúcej transkripcie po slovách čítaného toho istého textu. Práca je zhrnutím myšlienok a nápadov, ktoré som otestoval a popisujem aj úspešnosť ich použitia. Na záver som priložil zoznam výsledkov dosiahnutých pri testovaní.

Kľúčové slová: rozpoznávanie reči, dynamic time warping, DTW, slajdovacie DTW, preskakovacie DTW

Abstract

I have succesfully implemented the tool for solving problem of an automatic transcription the coherently read text with the knowledge of existing transcription of word by word read the same text. The thesis sumarizes the ideas and inspirations, which I also tested. I described the successfulness of their use, too. I attached the list of the reached results at the end of the thesis.

Key words: speech recognition, dynamic time warping, DTW, sliding DTW, overjumping DTW

Obsah

Kapitola 1

Úvod

Predstavme si budúcnosť. Staňme sa na chvíľku deťmi, využime našu fantáziu a prenesme sa o pár desaťročí do blízkej budúcnosti. Ako bude vyzerať svet? Ako budú vyzerať naše príbytky, autá, ... veci, ktoré každodenne používame? Určite budú plné všemožných technických vymožeností na predstavu ktorých treba dnes naozaj dobrú fantáziu. Jedna vec je ale očakávateľná a myslím, že každý ju dnes do svojej predstavy budúcnosti bez rozmýšľania zahrnul.

Ako ovládame všetky tie technické zázraky, ktoré nás obklopujú? Ako im oznamujeme, čo od nich chceme? Cez klávesnicu či iné "gombíkové" zariadenie? Príliš zdĺhavé a nemotorné. Ak si chceme užívať život v budúcnosti a netráviť väčšinu času stláčaním kláves, tak to určite z našej predstavy vyhodíme. Ovládanie pomocou myšlienok, pomocou nejakého, do nášho tela umelo pridaného zariadenia, alebo telepatie? To by sme asi zaradili až do výrazne vzdialenejšej budúcnosti. Čo mám teda na mysli? Aký je medzistupienok medzi týmito dvoma extrémami? Ak odpoveď ešte stále nie je jasná, skúste siahnuť po nejakom sci-fi filme prípadne knižke. Určite ju tam nájdete.

Komunikácia človeka s prístrojom pomocou prirodzenej ľudskej reči. To je správna odpoveď na vyššie uvedené otázky a tiež jeden z problémov, ktorého vyriešenie očakávame od blízkej budúcnosti.

Táto práca je venovaná riešeniu jedného z problémov patriaceho do tejto oblasti. Snažil som sa ju odľahčiť od prílišných technických detailov a jednotlivé veci vysvetľovať čo možno najzrozumiteľnejšie.

Prajem príjemné čítanie!

1.1 Trochu histórie

V dnešnej dobe si komunikáciu človeka so strojom väčšina ľudí spája s komunikáciou človeka s elektronickým počítačom. Možno sa budete diviť, ale tento problém začal človeka zaujímať dávno predtým, ako prvý elektronický počítač uzrel svetlo sveta. Na začiatku bola oblasť záujmu smerovaná skôr ku syntéze zvuku. V roku 1779 bol zostrojený prvý mechanický syntetizér reči, ktorý vedel syntetizovať zvuky samohlások. Toto zariadenie sa snažilo napodobovať ľudské hlasivky tak, že vháňalo prúd vzduchu cez zoskupenia rôznych prekážok, otvorov a štrbín, ktoré následne spôsobovali vznik zvuku [1].

Od tých čias sa daná oblasť neustále rozvíja a bolo vypracovaných množstvo postupov nápomocných či už pri syntéze, analýze alebo rozpoznávaní zvuku. Jedným z najvýznamnejších objavov bola diskrétna Fourierova transformácia, ktorej analógová verzia bola známa už v prvej polovici 18. storočia, ale svoje široké uplatnenie našla až s nástupom počítačov. Dôležitým momentom vývoja bol aj vynález spektrografu v roku 1946. Tento vynález povzbudil práce na analýze a syntéze reči, lebo umožňoval praktické a veľmi užitočné zobrazenie akustického signálu hlasového ústrojenstva.

Koncom 60-tych rokov boli prvý krát aplikované pri klasifikácii slov algoritmy založené na princípoch dynamického programovania. Táto metóda bola neustále vylepšovaná a dnes ju poznáme pod názvom dynamic time warping. Ďalšie výskumy viedli k objaveniu nových prístupov - modelovania pomocou skrytých Markovových modelov, alebo použitia neurónových sietí.

Dnes sú už známe a aj v praxi využívané rôzne systémy na rozpoznávanie hlavne izolovaných slov, pri ktorých je úspešnosť oveľa vyššia, ako pri dnešných rozpoznávačoch súvislej reči. Predpokladalo sa, že pri takomto tempe vývoja sa už čoskoro podarí skonštruovať rozpoznávač, ktorý bude kvalitatívne porovnateľný s ľudským sluchom. Dodnes tu však ostávajú niektoré otvorené problémy, ktoré zatiaľ len čakajú na svoje vyriešenie.

1.2 Aktuálne problémy

V čom teda spočíva kameň úrazu, že sa ho človeku nepodarilo uspokojivo vyriešiť ani za vyše pol storočia od vynálezu elektronických počítačov? Medzi hlavné problémy v oblasti rozpoznávania reči dnes patria [2]:

- Hlas každej osoby je iný. Muž, žena, dieťa, . . . , ale aj v rámci každej z týchto kategórií sa hlasy jedincov od seba viac či menej odlišujú. Je to spôsobené hlavne odlišnosťami v hlasovom ústrojenstve, frekvenciou kmitania hlasiviek či artikuláciou. Dôsledkom je, že každý človek má obvykle inú farbu hlasu, iný prízvuk, odlišné tempo reči a podobne. Na základe toho môže byť veľmi problematické zostrojiť rozpoznávač schopný správne interpretovať reč ľubovoľnej osoby.

- Ten istý človek nikdy nevysloví to isté slovo dvakrát rovnako. Rečový signál sa mení podľa hlasitosti vyslovovania. A keď do toho zarátame, že ráno, keď je človek rozospatý, prípadne chorý, alebo mu zabehlo, tak sa vyslovenie daného slova už nemusí ani zďaleka podobať tomu čo predtým. Významnú rolu tu hrá aj premenlivosť časovania. Časová dĺžka celého slova, ako aj časové dĺžky jeho jednotlivých častí, sa môžu výrazne líšiť od vyslovenia k vysloveniu podľa situácie a podmienok, za akých bolo slovo vyslovované.

- Na rečový signál má významný vplyv aj prostredie, v akom bola reč prednesená. V praxi sa nikdy nebude pracovať v ideálnych podmienkach absolútneho ticha. V pozadí sa neustále nachádzajú nejaké zdroje šumu, ktoré sa skladajú s daným signálom a spôsobujú problémy hlavne pri spracovaní slabých frikatív a pri detekcii začiatku a konca slova. Rozpoznávač preto musí byť schopný odlišovať, či signál zodpovedá hluku pozadia, alebo reči ktorú treba rozpoznať.

- Veľmi dôležitým problémom je aj to, že keď sú slová vyslovené súvisle v nejakom kontexte, tak nastáva jav koartikulácie - to znamená že sa môžu pozmeniť fonetické začiatky a konce slova v závislosti na kontexte okolitých slov.

- Logickou požiadavkou na rozpoznávače reči je práca online - teda spracovanie a rozpoznanie rečového signálu vo veľmi krátkom čase po prednesení reči. To prináša kopu nových problémov a stavia požiadavky nielen na presnosť použitých postupov, ale aj na ich rýchlosť. Tu môže významne dopomôcť aj vývoj hardvéru.

  Pre rozpoznávanie reči by bolo asi dobre, keby sa vývoj hardvéru uberal cestou napodobovania činnosti mozgu. Ten pravdepodobne pracuje ako

milióny paralelných procesorov s veľmi malou pamäťou. To je síce nevýhoda pri operáciách, kde treba robiť rýchle a presné výpočty a obyčajná kalkulačka je v porovnaní s mozgom výrazne lepšia. Pri rozpoznávaní nám ale počítač nie je rovnocenným súperom. Mozog dokáže veľmi rýchlo urobiť tisícky približných porovnaní počutého signálu so vzormi v pamäti. Rýchlo odfiltruje nepotrebnú informáciu a bez väčších problémov klasifikuje aj veľmi podobné vstupy [3].

- Je treba si uvedomiť aj to, že mozog a človek má oproti dnešným rozpoznávačom nepredstaviteľnú výhodu - on rozumie významu rozpoznávaného signálu. Pochopenie významu reči by tiež značnou mierou mohlo pomôcť pri klasifikácii hlavne súvislej reči.

1.3 Aký to má zmysel?

Aký to má celé význam? Načo riešiť tieto problémy, riešenia ktorých nám často aj dnes pripadajú ako niečo nepredstaviteľné? Je to z dôvodu obrovskej možnosti uplatnenia celej oblasti, či už syntézy reči, alebo jej rozpoznávania, v praktickom živote. Už dnes máme možnosť sa s nimi stretnúť.

Napríklad v oblasti nástrojov pre zrakovo postihnutých ľudí nájdeme mnoho pomôcok slúžiacich na uľahčenie ich života. Či sa už jedná o kalkulačky so syntetizovaným výstupom, ktorý používateľovi rečou oznámi výsledok, alebo o rôzne zariadenia reagujúce na slovné povely.

Ďalšou možnosťou uplatnenia dnes je napríklad rezervácia leteniek či objednanie jedla. Rozvíja sa oblasť automatizovaného prepisu správ a vytváranie zápisov zo súdnych pojednávaní. Rovnako oblasť tvorenia titulkov pre živé prenosy - kde trénovaný rečník hovorí komentár, ktorý sa ihneď divákom prepisuje ako textové titulky.

Rozpoznávače by mohli nájisť uplatnenie aj v školstve - pri výuke správneho hláskovania, alebo jazykov - pre správnu výslovnosť.

Vyššie uvedené veci sú len zlomkom z toho, kde všade by sa výsledky z danej oblasti mohli uplatniť.

1.4 Cieľ diplomovej práce

Jedná sa o experimentálno-implementačnú diplomovú prácu, ktorej cieľom je riešenie problému automatickej transkripcie súvislo prečítaného textu s využitím znalosti existujúcej transkripcie po slovách čítaného toho istého textu. Úlohou je navrhnutie a implementovanie algoritmu, ktorý na vstupe dostane zvukový signál - prirodzenou súvislou rečou prečítaný príbeh a ten istý príbeh prečítaný tým istým čitateľom slovo po slove s prestávkami (pričom budem vedieť, ktorá časť tohto signálu zodpovedá ktorému slovu). Na výstupe dostanem hranice jednotlivých slov v signále súvisle prečítaného príbehu. Algoritmus by mal fungovať aj na nekonzistentných vstupoch (napríklad ak je v jednom zo vstupných signálov slovo navyše) a zabrániť, aby nekonzistentnosť pokazila celý výpočet.

Tento problém nemusí byť riešený v reálnom čase (online). Cieľom je dosiahnutie čo najlepšieho času pri dostatočnej presnosti.

Takýto nástroj by sa dal v praxi použiť na obohatenie funkcionality už existujúcej Multimediálnej čítanky [4]. Jedná sa o výukový softvér slúžiaci deťom. Umožní im hravou a pútavou formou naučiť sa správnu výslovnosť slov slovenského jazyka. Po nahovorení príbehov v iných jazykoch by sa Multimediálna čítanka dala rovnako dobre použiť aj pri výučbe cudzích jazykov. Vytvorený nástroj by mal umožniť implementovať funkciu karaoke - po spustení prehrávania príbehu sú jednotlivé, práve čítané slová, zvýrazňované.

Motiváciou pre riešenie tohto problému okrem samotného faktu možnej praktickej užitočnosti výsledného produktu, je aj riešenie problému z oblasti rozpoznávania reči a obohacovanie tejto oblasti o výsledky mojich pokusov.

Na dosiahnutie uvedeného cieľa som mal k dispozícii dáta z projektu, v rámci ktorého slovenské základné školy a ich žiaci napísali a vyššie spomínanými spôsobmi nahovorili desiatky príbehov. Tieto príbehy som použil na testovanie môjho algoritmu.

1.5 Analýza problému

Na riešenie tohto problému sa ponúkajú dva prístupy. Prvým je použitie algoritmu Dynamic time warping (DTW) založenom na princípoch dynamického programovania a druhým použitie skrytých Markovových modelov (HMM). Ja

som si zvolil DTW, lebo si myslím, že mi lepšie umožňuje využiť fakt, že obidva vstupné zvukové súbory nahovoril ten istý rečník. Použitím HMM by som o túto výhodu prišiel, nakoľko by som iba zo vstupov nemal dostatočne veľkú množinu dát na ich natrénovanie, a musel by som použiť aj iné dáta. Úspešné použitie HMM by som si vedel predstaviť vtedy, ak by som mal jednotlivé izolované slová nahovorené tým istým rečníkom viackrát.

1.6 Čo bude nasledovať

V druhej kapitole oboznamujem čitateľa so základnými pojmami a technikami, ktoré sú už v danej oblasti známe a tým, čo už bolo v tejto oblasti skúmané. V tretej kapitole podrobnejšie popisujem algoritmus DTW, v štvrtej kapitole detailne rozoberiem svoje riešenie zadaného problému, v piatej sa venujem testovaniu a vyhodnoteniu dosiahnutých výsledkov a v šiestej zhrniem dosiahnuté výsledky a vyvodím záver. V prílohách na konci práce sú umiestnené konkrétne výsledky pre dva testovacie vstupy, zoznam parametrov používaných algoritmom a ich významy a nachádzajú sa tam aj informácie o priloženom CD a jeho obsahu.

Kapitola 2

Prehľadová kapitola

V tejto kapitole sa budem venovať akémusi všeobecnému prehľadu. Od základných definícií, ktoré objasnia, čo to reč je a akú informáciu pre nás predstavuje, prejdem postupne k štandardným postupom ako ju spracovávame a následne spomeniem, aké metódy rozpoznávania reči sa dnes najviac uplatňujú.

Sústredil som sa hlavne na tie témy, ktoré som priamo, alebo nepriamo použil aj pri riešení problému tejto diplomovej práce. Snažil som sa čo najjasnejšie vystihnúť ich podstatu. Záujemcov o podrobnejšie vysvetlenie spomínaných pojmov odkazujem na [2], kde sa dá nájsť detailne popísaná teória väčšiny z toho, čo bude spomenuté.

2.1 Reč a jej obsah

Na začiatok si povieme čo to vlastne reč je a aký má pre nás informačný obsah.

2.1.1 Od zvuku k reči

Zvuk je pozdĺžne mechanické vlnenie v látkovom prostredí, ktoré pôsobí na sluchový orgán a vyvoláva sluchový vnem. Frekvencia tohto vlnenia sa pohybuje v rozmedzí 16 Hz až 20 KHz. Mimo týchto hraníc človek zvuk nevníma. V širšom zmysle sa dá považovať za zvuk aj vlnenie mimo týchto hraníc - jedná sa o infrazvuk a ultrazvuk. Jednotkou hlasitosti zvuku (akustického tlaku) je decibel (dB). Zvuk sa šíri od zdroja v guľových vlnoplochách meniaceho sa tlaku, čo spôsobuje lokálne stláčanie a roztiahovanie jednotlivých lokálnych kúskov hmotného prostredia. Čiastočky prostredia sú "nahradené" vlnou a oscilujú. S rastúcou vzdialenosťou slabne energia tohto kmitania - o 6 dB pri zdvojnásobení vzdialenosti. Hladina akustického tlaku pri bežnom rozhovore je asi 30 dB. Pri kriku asi 80 dB. Veda ktorá sa zaoberá zvukom sa nazýva akustika.

Hlas je zvukový prejav ľudí vytváraný hlasovými orgánmi. Základný tón vzniká v hrtane rozkmitaním hlasiviek prúdom vzduchu z pľúc. Následne sa zosilňuje sústavou rezonančných dutín. Frekvencia ľudského hlasu sa štandardne pohybuje v rozmedzí približne 64 Hz až 1 024-2 048 Hz

Reč je hlasom kódovaná informácia slúžiaca na komunikáciu medzi rečníkom a poslucháčom. Pravidlá kódovania informácie do hlasu sú určené príslušným jazykom, v ktorom komunikácia prebieha.

2.1.2 Informačný obsah fonetickej formy

Najmenšou človekom stanovenou jednotkou reči je fonéma. Fonémy môžeme od seba odlíšiť podľa miesta ich vzniku v rečovom trakte, alebo podľa sluchového dojmu. Výskumy na svetových jazykoch ukázali že exituje akýchsi 12 univerzálnych diferenčných príznakov a s nimi spojených 12 polôh hlasového traktu človeka, ktorých postupnosti vytvárajú jednotlivé fonémy.

Pre dnes používané jazyky sa pohybuje počet foném v nich používaných od 12 do 60. v anglickom jazyku ich je 42, v ruskom 40 a v slovenčine 36. Vyššou stavebnou jednotkou reči je slabika. Sú tvorené spájaním foném presne určenými pravidlami. Ďalšími jednotkami sú slová - tvorené postupnosťami slabík

podľa pravidiel príslušného jazyka, a vety - zoskupenia slov. Slovanské jazyky používajú približne 2 500 - 3 500 slabík a 45 000 - 50 000 slov.

A ako sa dá teda popísať množstvo informácie obsiahnutej v reči? Pre jednoduchosť predpokladajme, že na tvorbu reči máme k dispozícii 32 foném. Potom priemerná informácia pripadajúca na jednu fonému je

$$\mathrm{I} = \log_2 32 = 5 \quad (\mathrm{bitov})$$

Táto hodnota by bola pri skutočnom jazyku nižšia. Keď budeme brať do úvahy relatívne pravdepodobnosti výskytu jednotlivých foném, tak nám táto hodnota klesne asi na 4,5 bit (niečo ako kompresia na princípe častejšie sa vyskytujúceho znaku). A skúmaním určitých jazykových pravidiel - štatistických vlastností rôznych kombinácií fonetických postupností, nám klesne priemerná informácia pripadajúca na jednu fonému až na približne 3-3,5 bit.

Pri bežnom rozhovore človek vysloví asi 80-130 slov za minútu, čo zodpovedá asi 10-tim fonémam za sekundu. Ak je teda priemerná informácia každej z nich 3-3,5 bit, informácia obsiahnutá v hovorenej reči bude 30-35 bit/s. Tento odhad je v súlade s psychoakustickými testami, podľa ktorých je človek schopný spracovávať sluchovú informáciu o rýchlosti maximálne 50 bit/s.

2.1.3 Informačný obsah akustickej formy

Ak chceme pracovať s kvalitným signálom hovorenej reči musíme brať do úvahy, že vďaka vysokému frekvenčnému rozsahu frikatív dosahuje najvyššie kmitočty až okolo 10 kHz. Keď si vezmeme Shannonovu teorému (podľa ktorej spojitú funkciu času s hornou hranicou kmitočtového spektra F_m , možno nahradiť postupnosťou jej diskrétnych hodnôt zosnímaných s frekvenciou $F_v \geq 2F_m$), tak vzorkovacia frekvencia F_v (koľkokrát za sekundu zaznamenám signál) pre zachovanie kvality signálu musí byť aspoň 20 kHz. Ak na každú vzorku potrebujem 8-12 bitov tak výsledná rýchlosť prenosu informácie bude ≈200 000 bit/s.

V porovnaní s výsledkami predchádzajúcej časti, kde je odhadnutá veľkosť informácie fonetickej formy, teda vidíme obrovskú informačnú redundanciu akustického signálu, ktorý prenáša rovnakú rečovú informáciu ako príslušná postupnosť foném. Toto je spôsobené tým, že akustický signál nesie okrem samotnej informácii o reči aj informácie o intonácii, tempe reči, farbe hlasu, dialekte, prípadnými defektami reči rečníka a podobne.

Ľudský sluchový systém pri vnímaní akustického signálu pracuje tak, že potláča informácie, ktoré nie sú potrebné na práve vykonávanú činnosť. To znamená, že keď sa snažíme porozumieť tomu čo rečník hovorí, tak sú všetky prebytočné informácie o tempe, dialekte, farbe hlasu... odfiltrované. Keď má byť ale výsledkom vnímania reči napríklad identifikácia rečníka, tak sa využijú zase iné charakteristiky signálu.

2.2 Spracovanie signálu

Pri otázke voľby spracovania rečového signálu treba uvážiť, aký je cieľ spracovania. Iná reprezentácia sa použije pre výber informatívnych príznakov a iná pre efektívne kódovanie. Pri výbere vhodnej reprezentácie reči nás zaujímajú hlavne:

- Zložitosť - je určená množstvom matematických operácií potrebných na získanie zvolenej reprezentácie signálu.
- Rýchlosť prenosu informácie - dáva nám určitý údaj o redundancii daného signálu. Príliš malá znamená možnosť efektívneho zapamätávania a prenosu, ale za cenu možnej redukcie informačného obsahu.
- Pružnosť - určuje ako dobre a efektívne sa so zvolenou reprezentáciou dá ďalej pracovať.

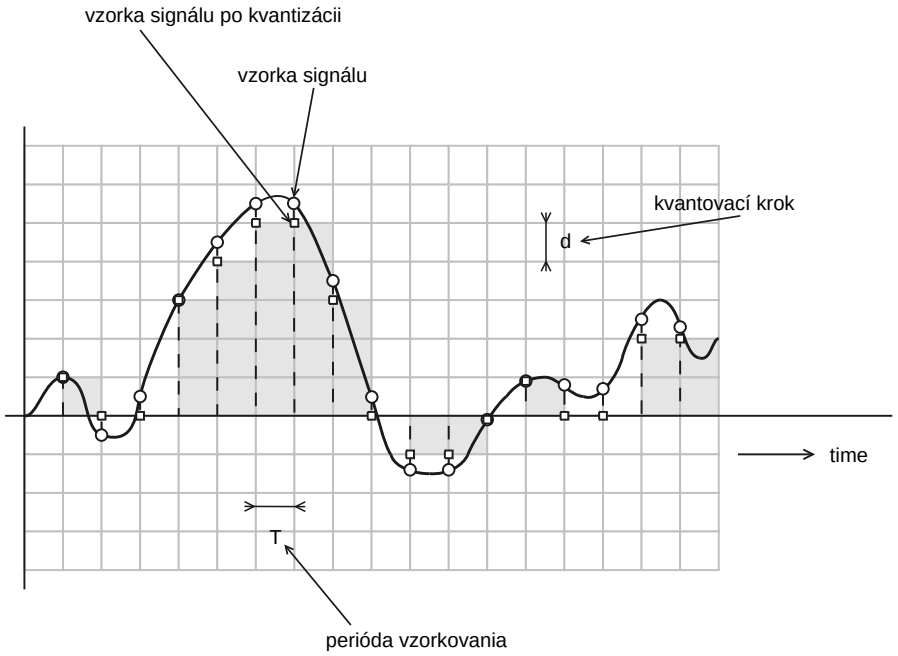
Pri väčšine metód spracovania signálu sa vychádza z predpokladu, že sa jeho vlastnosti v priebehu času menia pomaly. Vychádza to z výskumov rečového traktu človeka, ktoré ukázali, že jeho zotrvanie v jednotlivých stavoch trvá určitý krátky čas. To viedlo k tomu, že tieto metódy pracujú na princípe krátkodobej analýzy, kde sa s malými úsekmi signálu pracuje, akoby to boli krátke zvuky. Tieto úseky sa nazývajú mikrosegmenty a popisujú kúsok signálu dlhý spravidla 10-30 ms. Výsledkom analýzy je potom číslo alebo vektor popisujúci daný kúsok signálu. Mikrosegmenty na seba nadväzujú, alebo sa môžu prekrývať, a tým pádom nimi môžem popísať ľubovolne dlhý kus signálu.

2.2.1 Pulse Code Modulation (PCM)

Rečový signál ako taký má analógový charakter. Na jeho zaznamenávanie sa obvykle používa mikrofón a následne je potrebné previesť analógový signál

na digitálny, aby sme s ním mohli d'alej pracovať. Zariadenie, ktoré to sprostredkúva, sa nazýva A/D[1] prevodník a na jeho realizáciu sa štandardne používa metóda pulznej kódovej modulácie.

Funguje to tak, že rozdelíme časový priebeh signálu na krátke úseky a na začiatku každého z nich zaznamenáme hodnotu signálu. Názorne to môžeme vidieť na obrázku 2.1.



Obrázok 2.1: PCM

Pre korektné zaznamenanie zvuku musíme dodržať určité kritériá. Jedným z nich je vzorkovanie. Pri rozhodovaní, ako často (s akou periódou T) budeme hodnoty signálu zaznamenávať, musí byť dodržaná Shannonova vzorkovacia teoréma - frekvencia vzorkovania $F_v = \frac{1}{T}$ musí byť aspoň 2-krát taká, ako je horná hranica frekvenčného rozsahu zaznamenávaného signálu ($F_v \geq 2F_m$). v opačnom prípade dochádza k skresleniu zložiek vyšších frekvencií.

Kvantizácia a následné kódovanie je aproximácia analógovej hodnoty vzorky signálu na hodnotu z konečnej množiny číselných hodnôt. Určí sa počet pásiem (obvykle v tvare 2^B , kde B je počet bitov v binárnom kóde), na ktoré rovnomerne rozdelíme rozsah signálu a jednotlivé hodnoty signálu zaokrúhľujeme na hod-

[1] Analog/Digital

notu najbližšieho pásma. Kvantizáciou signálu dochádza k určitej strate informácie, ktorej veľkosť závisí od počtu pásiem.

Prenosová rýchlosť informácie sa dá potom vyjadriť ako $F_v B$ bit/s.

Okrem základnej PCM sa často používajú aj rôzne jej modifikácie, ktorých cieľom je zníženie rýchlosti prenosu informácie pri zanedbateľnej strate informačného obsahu. Patria sem napríklad:

- Nelineárne kódovanie - Lineárne rozloženie pásiem pri kvantizácii sa ukázalo ako nie príliš efektívne. Pri znelých (vyššie amplitúdy) segmentoch reči nie je potrebné mať až taký malý kvantizačný krok zatiaľ čo pri neznelých (nižšie amplitúdy) je na ich presné zaznamenanie dobré mať kvantizačný krok čo najmenší. Na základe empirických zistení rozložení amplitúd v rečovom signále sa ukázalo, že práve logaritmická charakteristika kvantizéru je pre tento účel skoro ideálna. Pre porovnanie - pri logaritmickom PCM s rozsahom 12 bit/vzorku dostávam kódovanie približne rovnakej kvality ako pri lineárnom PCM s rozsahom 16 bit/vzorku.

- Diferenčná PCM (DPCM) - Vychádza z pozorovaní že susedné vzorky pri PCM analýze sa od seba príliš neodlišujú. To znamená že rozdiely susedných vzoriek budú mať menšiu disperziu ako samotné vzorky. A teda si namiesto vzorky samotnej budeme zapamätávať jej rozdiel od predchádzajúcej, čím znížime prenosovú rýchlosť.

Výstup metódy digitalizácie zvuku využívajú ako vstup ostatné metódy krátkodobej analýzy signálu. Pri ďalších spomínaných metódach budeme preto pod signálom rozumieť signál spracovaný pomocou nejakého kódovania vstupnej vlny (napr. PCM).

2.2.2 Funkcia okienka

Jedná sa o jednoduchý nástroj, ktorý sa aplikuje na signál predtým, ako sú aplikované ďalšie postupy spracovania. Úlohou okienka je vybrať príslušné vzorky s_n kde $n = 0, \ldots, N-1$ spracovávaného mikrosegmentu dĺžky N a vynásobiť ich váhou w_n. Pre pravouhlé okienko je váha $w_n = 1$. Pre Hammingovo okienko je definovaná vzťahom:

$$w_n = 0,54 - 0,46 \cos \frac{2\pi n}{N-1}$$

To spôsobí že hodnoty sú utlmované smerom k okrajom mikrosegmentu, zatiaľ čo v strede ostávajú zachované. Spravidla sa tým dosahujú lepšie výsledky spracovania signálu ako pri pravouhlom okienku.

2.2.3 Krátkodobá energia a krátkodobá intenzita

Krátkodobú energiu signálu zodpovedajúcu k-temu mikrosegmentu dĺžky N vypočítame vzťahom:

$$E_k = \sum_{n=0}^{N-1} (s_n w_n)^2$$

kde s_n je n-tá hodnota signálu príslušného mikrosegmentu a w_n je nešpecifikovaná funkcia okienka. Hodnoty pre jednotlivé mikrosegmenty poskytujú informácie o energii signálu v týchto mikrosegmentoch. Problémom pri energii je veľká citlivosť na veľké zmeny úrovne signálu a dynamiku signálu ešte zväčšuje druhá mocnina. Preto býva veľmi často namiesto funkcie krátkodobej energie používaná funkcia krátkodobej intenzity:

$$M_k = \sum_{n=0}^{N-1} (|s_n| w_n)$$

ktorá tento nedostatok už nemá. Krátkodobá energia aj krátkodobá intenzita sa dajú využívať na hľadanie segmentov ticha a segmentov reči, pomocou čoho je možné rozdeľovať signál na bloky reči oddelené tichom. Hodnoty funkcie energie sa tiež používajú ako príznaky v jednoduchých klasifikátoroch slov.

2.2.4 Diskrétna Fourierova Transformácia (DFT)

Postupnosť hodnôt $s_0 \dots s_{N-1}$ určitého mikrosegmentu (už upraveného príslušnou, bližšie nešpecifikovanou funkciou okienka) dĺžky N signálu upravím pomocou nasledujúceho predpisu:

$$S_k = \sum_{n=0}^{N-1} (s_n e^{-\frac{2\pi i}{N} kn}) \qquad k = 0, \dots, N-1$$

kde:

$$e^{\frac{2\pi i}{N}} = \cos\frac{2\pi}{N} + i \sin\frac{2\pi}{N}$$

a dostanem postupnosť hodnôt $S_0 \dots S_{N-1}$ komplexných čísel, kde e je eulerova konštanta a i je imaginárna jednotka. Táto operácia sa nazýva krátkodobá

diskrétna Fourierova transformácia a získame ňou Fourierove koeficienty. Fourierova transformácia premietne signál do frekvenčného a fázového spektra. Spektrum je rovnomerne rozdelené na N frekvencií a reálne zložky výsledných koeficientov nám hovoria o veľkostiach týchto frekvencií - ako sú zastúpené v pôvodnom signále. Imaginárna zložka nám zase hovorí o veľkostiach fázových posunov. Oproti pôvodnému signálu sme veľmi nezmenili informačnú redundanciu, ale poskytuje nám možnosť odfiltrovať požadované frekvencie a skúmať spektrálne vlastnosti rečového signálu na výstupe úzkych frekvenčných pásiem. Rovnako máme možnosť z Fourierovych koeficientov inverznou transformáciou rekonštruovať pôvodný signál podľa vzťahu:

$$s_n = \frac{1}{N} \sum_{k=0}^{N-1} (S_k e^{\frac{2\pi i}{N} kn}) \qquad n = 0, \ldots, N-1$$

Výpočet DFT podľa uvedeného vzorca vyžaduje v priemere N^2 operácií súčtu komplexných čísel, čo by mohlo predstavovať problém pri systémoch spracovania fungujúcich v reálnom čase. Našťastie je známy algoritmus Rýchlej Fourierovej Transformácie - Fast Fourier Transformation (FFT). Funguje dobre za podmienky že dĺžka mikrosegmentu je rovná 2^m pre nejaké $m \in \mathrm{N}$. Využíva sa v nej periodicita $e^{-\frac{2\pi i}{N} kn}$ a dovoľuje znížiť počet nutných operácií násobenia až na hodnotu $\frac{1}{2} N \log_2 N$.

2.2.5 Mel Frequency Cepstral Coefficients (MFCC)

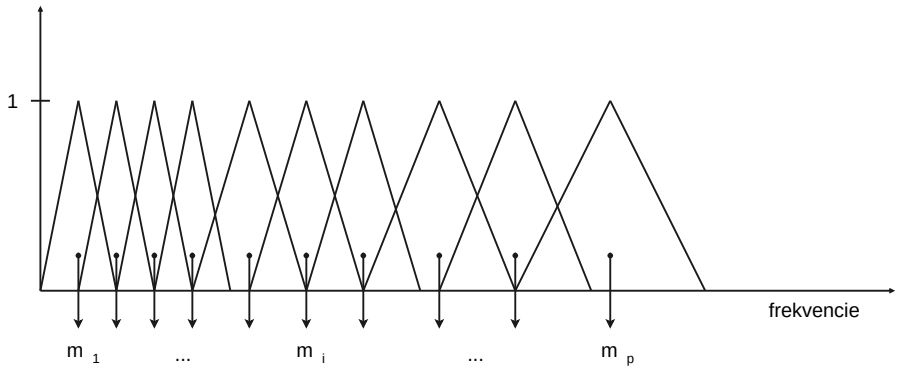
MFCC sú dnes asi najčastejšia reprezentácia signálu používaná pri rozpoznávaní reči. Svedčia o tom viaceré práce, ktoré sa zaoberali skúmaním jednotlivých metód a výsledkov dosiahnutých pomocou nich. Viac sa o porovnávaní jednotlivých reprezentácii môžete dočítať napríklad v prácach [5] a [6], ktoré sa zaoberajú danou problematikou.

Hlavnou črtou MFCC je snaha upraviť signál tak, ako ho asi vníma ľudské ucho. Ľudské ucho je špecifické tým, že nevníma jednotlivé frekvencie zvukového signálu lineárne - v nižších frekvenčných pásmach je citlivejšie, zatiaľ čo vo vyšších menej. Tento prístup zohľadňujeme pri tvorbe MFCC a dosahujú sa tak spravidla lepšie výsledky. Pri spracovávaní sa to rieši zavedením Mel škály, ktorá odlinearizuje pôvodné frekvencie. Konverziu frekvencie f v Hz na Mely určuje predpis:

$$m = 2595 \log_{10} (1 + \frac{f}{700}) [Mel]$$

Celý postup vypočítania MFCC vyzerá nasledovne:

1. Z vstupného mikrosegmentu najprv pomocou FFT vypočítam Fourierove koeficienty (FC).
2. Z FC vypočítam amplitúdové spektrum pomocou vzťahu $\sqrt{a^2 + b^2}$, kde a a b sú reálna a imaginárna zložka príslušného koeficientu získaného z FT.
3. Vytvorí sa banka filtrov zodpovedajúca Mel škále - čím nižšia frekvencia, tým vyššia citlivosť (do filtra vstúpi užší rozsah frekvencií). Počet filtrov p je určený používateľom a určuje počet frekvenčných pásiem. Štandardne sa používa hodnota 20 až 40. Názorne to môžeme vidieť na obrázku 2.2.



Obrázok 2.2: Mel škála - banka filtrov

4. Banka filtrov sa aplikuje na amplitúdové spektrum - vynásobíme jednotlivé hodnoty amplitúdového spektra zložkami filtra pre príslušný kanál a sčítame. Tak dostaneme amplitúdy m_i kde $i = 1, \ldots, p$ zodpovedajúce príslušným kanálom.
5. MFCC následne vypočítame kosínusovou transformáciou:

$$c_k = \sqrt{\frac{2}{p}} \sum_{i=1}^{p} \left(m_i \cos(\frac{\pi i}{p}(i - 0,5)) \right)$$

2.3 Rozpoznávanie reči

Pod rozpoznávaním vo všeobecnosti rozumieme zaraďovanie jednotlivých rozpoznávaných entít do vhodných tried. Pri reči sú tieto entity predstavované fonémami, prípadne vyššími jednotkami (napr. slovami).

Rozpoznávanie reči je komplexný problém. Počnúc analýzou signálu a jeho transformáciou do nejakej vhodnejšej reprezentácie (napríklad MFCC), kategorizovaním do tried a končiac syntaktickou a sémantickou analýzou.

Vhodnejšiu reprezentáciu signálu nazývame príznaky a množinu rozpoznávaných tried pri rozpoznávaní reči nazývame slovník. Pred samotným rozpoznávaním je nutné tento slovník vytvoriť a mať ho vhodne reprezentovaný v pamäti, aby sa s ním dalo efektívne pracovať.

Dnešné metódy rozpoznávania reči sa dajú podľa použitých postupov rozdeliť na tri základné smery:

2.3.1 Dynamic Time Warping (DTW)

Táto metóda je založená na princípe dynamického programovania. Vychádza z toho, že pri vyslovovaní toho istého slova sa stáva, že jeho jednotlivé časti majú od vyslovenia k vysloveniu inú dĺžku trvania. Napríklad slovo "nie" môžem vysloviť ako "nie", "niééé" či "níííééé" prípadne nejakú jeho časť kvôli neočakávanému vyrušeniu nevysloviť. Snažíme sa teda prostredníctvom nelineárnej transformácie časovej osy - predlžovaním, prípadne skracovaním jednotlivých mikrosegmentov z ktorých je slovo zložené, čo najlepšie namapovať príznaky vstupu na slová zo slovníka a vyberáme, to kde sme dosiahli čo najlepšiu zhodu. Nevýhodou je, že je to použiteľné len pre menšie slovníky. Pri väčších to môže byť časovo veľmi náročné. Keďže som algoritmus DTW použil ako základné východisko pre túto prácu, celá nasledujúca kapitola je venovaná práve jemu. Sú v nej detailne popísané jeho rôzne podoby a formy.

2.3.2 Skryté Markovove Modely (HMM)

Jedná sa o štatistickú metódu založenú na princípoch vytvárania reči človekom (o ktorých je známe v súčasnosti viac ako o rozpoznávaní reči človekom). Základný princíp je taký, že ku každému slovu zo slovníka sa najprv zostrojí skrytý Markovov model - konečný automat s pravdepodobnostnými prechodmi, kde v každom stave máme určené pravdepodobnosti zodpovedajúce jednotlivým zvukom tvoriacich slovo. V jednotlivých modeloch pre konkrétny vstup hľadám cestu s najväčšou pravdepodobnosťou vydania danej postupnosti zvukov a zvíťazí slovo, v ktorého modeli sa táto cesta našla. V praxi sa nepoužívajú modely pre jednotlivé slová, ale pre fonémy a tie sa určitými postupmi spája-

jú do slov. Učenie sa týchto modelov je časovo pomerne náročné, ale samotné rozpoznávanie už prebieha rýchlo.

Aj keď by bolo za určitých predpokladov možné riešiť problém tejto diplomovej práce pomocou HMM, ja som sa rozhodol pre inú cestu, a preto sa im podrobnejšie už nebudem venovať. Prípadný záujemca sa môže o nich dočítať v [2], kde je detailne popísaná ich teória, prípadne v práci [7] sústredenej na využitie HMM pri rozpoznávaní číslic a tiež prácach [8] a [3].

2.3.3 Neurónové siete

Tak ako pri mnohých problémoch kde sa neurónové siete dobre uplatnili, aj pri rozpoznávaní reči ľudí napadlo využiť ich schopnosť všímať si podobnosti a na základe nich kategorizovať vstupy do tried. Pri rozpoznávaní sa nastavia jednotlivé parametre neurónovej siete trénovaním na známych vstupoch. Sieť potom dokáže asociovať, a tak správne klasifikovať aj iné vstupy. Rozpoznávaním reči pomocou neurónových sietí sa zaoberá práca [9], kde sa zaoberali porovnávaním viacerých typov neurónových sietí na rozpoznávanie izolovaných slov. Podobne aj práca [10] sa zaoberá rozpoznávaním izolovaných slov pomocou rekurentnej neurónovej siete a úspešnosť je porovnateľná s úspešnosťou systémov fungujúcich na báze HMM.

Kapitola 3

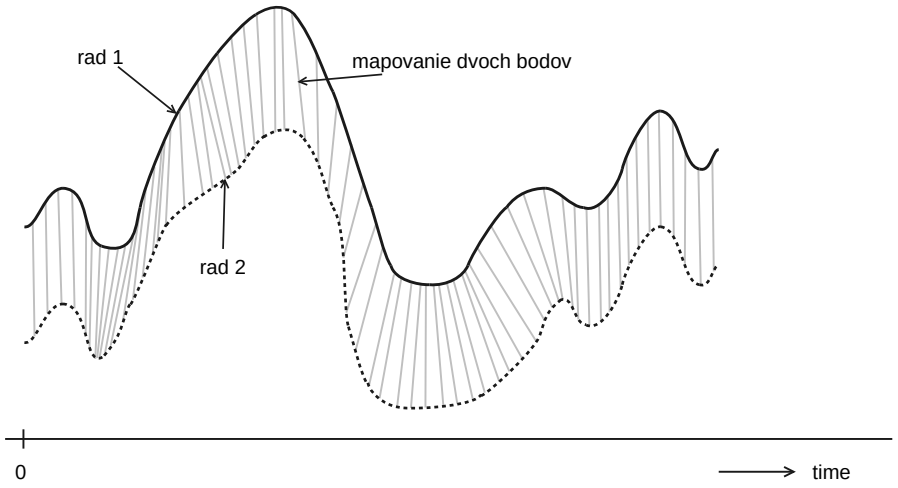
Dynamic Time Warping

Časové rady (postupnosti) sú dnes veľmi často používanou formou dát zastúpenou v mnohých vedeckých oblastiach. Častým problémom pri časových radoch je určovanie, ako veľmi sú si dva rady navzájom podobné. V niektorých prípadoch sa to dá riešiť jednoduchým porovnaním po dvojiciach jednotlivých členov radov napríklad pomocou Euklidovej vzdialenosti. Často sa však stáva, že tieto postupnosti majú približne rovnaký priebeh, iba sú celé, alebo ich časti, navzájom trochu posunuté vzhľadom na časovú os. Keď chceme nájsť podobnosť medzi takýmito dvomi postupnosťami, musíme najprv poohýbať časovú os jednej alebo obidvoch postupností predtým, ako ich po jednotlivých prvkoch porovnáme. Práve týmto problémom sa zaoberá algoritmus DTW.

V tejto kapitole podrobne popisujem spomínaný algoritmus, princíp jeho fungovania, obmedzenia a heuristiky, ktoré napomáhajú dosahovaniu lepších výsledkov pri jeho aplikovaní, jeho implementáciu a jeho rôzne používané formy hlavne z oblasti rozpoznávania reči.

3.1 Základné myšlienky a pojmy

Predpokladajme, že máme dve postupnosti. Pri akustickom signále sú to postupnosti koeficientov alebo vektorov príznakov, získané pomocou nejakej z metód predspracovania signálu, alebo kombináciou týchto metód. Každá z týchto postupností môže predstavovať napríklad vyslovenie nejakého slova. Naším cieľom je zistiť, či sa jednalo o to isté slovo. Pomocou nelineárnej transformácie časovej osi treba nájsť správne namapovanie jednotlivých častí týchto postupností a porovnať ich medzi sebou. Názorne to pre postupnosti s jednorozmernými prvkami môžeme vidieť na obrázku 3.1.



Obrázok 3.1: Vzájomné namapovanie dvoch radov na seba pomocou ohýbania časovej osi

Označme si tieto postupnosti

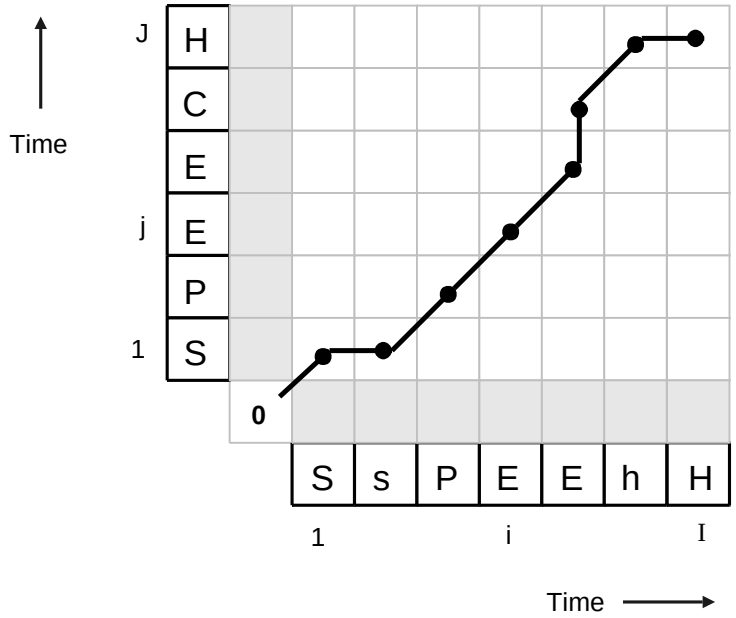
$$A = \{a_1, a_2, \ldots, a_n, \ldots, a_I\}$$

$$B = \{b_1, b_2, \ldots, b_m, \ldots, b_J\}$$

kde x_i je i-ty vektor príznakov príslušnej postupnosti. Na porovnanie týchto dvoch postupností pomocou algoritmu DTW, skonštruujeme I x J maticu vzdialeností M prislúchajúcu postupnostiam A a B, kde na mieste $M(i, j)$ bude vzdialenosť (napríklad euklidovská) $d(a_i, b_j)$ vektorov a_i a b_j. Ohýbacia cesta C, určujúca namapovanie postupnosti A na postupnosť B, bude podmnožina

z prvkov M, kde k-ty prvok cesty C bude definovaný ako $c_k = M(i_k, j_k)$, kde k je časová premenná a i_k a j_k sú príslušné vektory príznakov zodpovedajúce času k. Príklad ohýbacej cesty je znázornený na obrázku 3.2.

$$C = c_1, c_2, \ldots, c_k, \ldots, c_K \qquad max(I, J) \leq K \leq I + J - 1$$



Obrázok 3.2: Príklad ohýbacej cesty - porovnanie signálov dvoch možných vyslovení slova ”speech”. Pre zjednodušenie sa predpokladá, že každé písmenko je reprezentované jednou hodnotou (vektorom).

Ohýbacia cesta C musí ešte spĺňať nasledovné podmienky:

- **Hraničné body:**

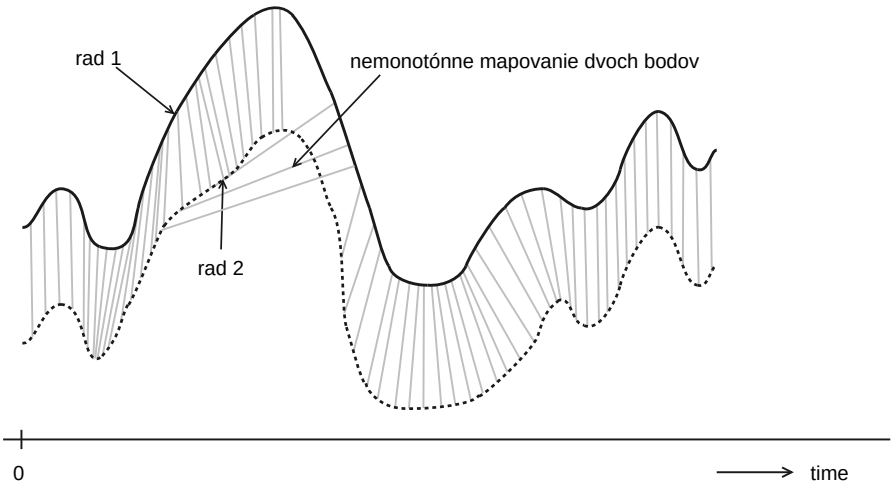
  Určujú začiatok a koniec ohýbacej cesty C. V prípade hľadania najlepšieho namapovania celej postupnosti A na postupnosť B (spracovanie izolovaných slov) môžeme toto obmedzenie vyjadriť podmienkami:

$$c_1 = (1, 1)$$

$$c_K = (I, J)$$

- **Monotónnosť:**

  Nie je prípustné, aby sa časť jednej postupnosti namapovala viackrát na tú istú časť druhej postupnosti, ak tieto časti nenasledujú priamo za sebou. Pre postupnosť s jednorozmernými prvkami je to znázornené na obrázku 3.3.



Obrázok 3.3: Neprípustné nemonotónne namapovanie dvoch radov

Tomuto javu zabraňujeme podmienkami zaručujúcimi monotónnosť:

$$0 \leq i_k - i_{k-1}$$

$$0 \leq j_k - j_{k-1}$$

- **Lokálne obmedzenia - kontinuita a strmosť:**

  Na zabezpečenie, aby nedochádzalo k nadmernej kompresii, či expanzii časových osí porovnávaných postupností, aplikujeme na ohýbaciu cestu C nasledovné podmienky:

$$i_k - i_{k-1} \leq I^*$$

$$j_k - j_{k-1} \leq J^*$$

  V praxi volíme $I^*, J^* = 1, 2, 3$. Ak pripustíme hodnoty vyššie ako 1, znamená to, že pri namapovaní sa môžu niektoré prvky postupností vynechať.

Nebude dobré, ak bude viesť ohýbacia cesta príliš dlho v jednom smere, zatiaľ čo v druhom nie. Znamenalo by to, že sa krátky úsek jednej postupnosti namapoval na veľmi dlhý úsek druhej postupnosti. Ak tomu chceme zabrániť, musíme mať zabezpečenú podmienku obmedzujúcu strmosť. Mohla by vyzerať napríklad takto:

Ak na ceste C idem x-krát po sebe v smere jednej z osí, tak cesta v tomto smere nemôže ďalej pokračovať, pokým sa nespraví y-krokov v inom smere (napríklad pozdĺž diagonály).

- **Globálne obmedzenia:** Ak zovšeobecníme lokálne podmienky strmosti ohýbacej cesty C na jej celkový priebeh, je možné pri splnení podmienok určujúcich hraničné body vymedziť prípustnú oblasť jej prechodu maticou M:

$$1 + \alpha(i_k - 1) \leq j_k \leq 1 + \beta(i_k - 1)$$

$$J + \beta(i_k - I) \leq j_k \leq J + \alpha(i_k - I)$$

kde α a β sú smernice priamok vymedzujúce prípustnú oblasť, ktoré sú odvodené z lokálneho obmedzenia cesty C.

Ďalej môžeme predpokladať, že pri porovnávaní dvoch postupností prislúchajúcich vysloveniu toho istého slova, nebude mať kolísanie tempa reči za následok zásadné časové rozdiely príslušných úsekov oboch postupností. Preto by malo platiť:

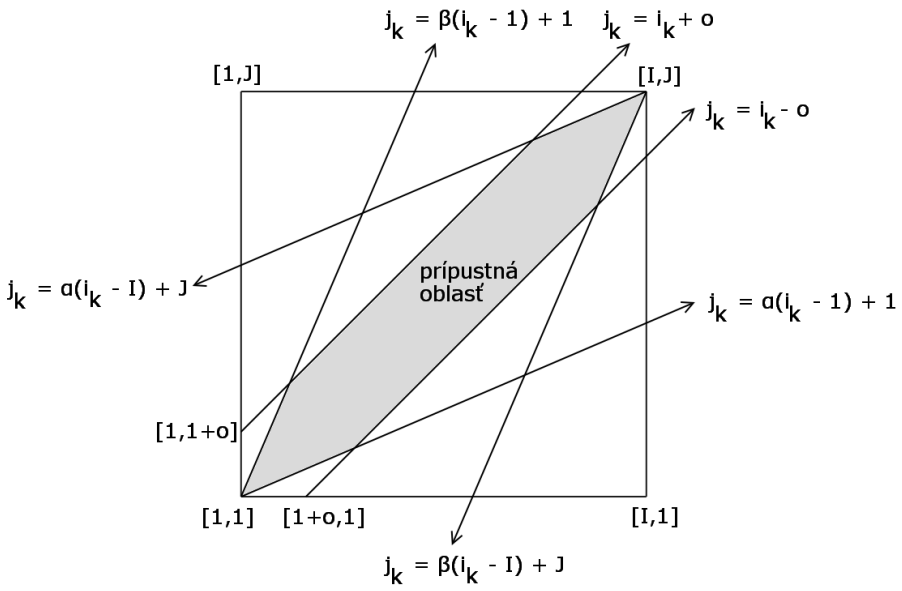
$$|\frac{J}{I} i_k - j_k| \leq o$$

kde o je vhodné celé číslo. Intuitívne si pod tým môžeme predstaviť, že prvok jednej postupnosti a prislúchajúci prvok druhej postupnosti vzhľadom k pomeru celkových dĺžok postupností, nebudú od seba vzdialenejšie, ako nejaká zvolená hranica o.

Obidva prípady globálnych ohraničení sú znázornené na obrázku 3.4.

Pre dve postupnosti existuje exponenciálne veľa ohýbacích ciest spĺňajúcich tieto podmienky. Nás ale bude zaujímať len tá, ktorá minimalizuje ohýbaciu cenu - vzájomnú vzdialenosť postupností po poohýbaní ich časových osí. Túto cestu ako minimum cez všetky ohýbacie cesty a jej cenu počíta algoritmus DTW:

$$DTW(A, B) = \min \frac{\sum_{k=1}^{K} c_k W(k)}{N(W)}$$



Obrázok 3.4: Globálne ohraničenia ohýbacej cesty

kde c_k je príslušný prvok z ohýbacej cesty C, $W(k)$ je hodnota váhovej funkcie pre k-ty úsek cesty C a $N(W)$ je normalizačný faktor, ktorý je funkciou váhovej funkcie.

Ešte ostáva vysvetliť pojmy vzdialenosť, váhová funkcia a normalizačný faktor:

- **Vzdialenosť:**

  Lokálna vzdialenosť a jej výpočet je závislý na konkrétnych typoch dát v postupnostiach. Vo veľa prípadoch použitia DTW si vystačíme zo štandardnou euklidovskou vzdialenosťou:

  $$d_{eukl}(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

  kde x a y sú vektory dĺžky n. Pre MFCC koeficienty z ich štatistických vlastností vyplýva, že nie každá súradnica by sa mala na celkovej vzdialenosti podieľať rovnakou váhou. To umožňuje tzv. Mahalanobisova miera d_{MCEP} definovaná vzťahom:

  $$d_{MCEP}(x,y) = (x-y)V^{-1}(x-y)^T$$

kde x a y sú MFCC vektory, T značí transponovaný vektor[1] a V je kovariačná matica určená pomocou MFCC koeficientov. Keďže je ale výpočet tejto miery časovo dosť náročný, často sa nahradzuje jednoduchšou mierou d_{WCEP}:

$$d_{WCEP}(x, y) = \sum_{i=1}^{n} v_i(x_i - y_i)^2$$

kde x a y sú MFCC vektory dĺžky n a v_i je inverzný prvok i-teho diagonálneho prvku kovariačnej matice V.

- **Váhová funkcia:**

  Akú váhovú funkciu použijeme pre konkrétny prípad závisí iba od použitých lokálnych obmedzení ohýbacej cesty. Medzi najčastejšie používané typy váhových funkcií patria:

  - typ a) symetrická váhová funkcia

    $$W(k) = (i_k - i_{k-1}) + (j_k - j_{k-1})$$

  - typ b) asymetrická váhová funkcia
    - b1)

      $$W(k) = i_k - i_{k-1}$$

    - b2)

      $$W(k) = j_k - j_{k-1}$$

  - typ c)

    $$W(k) = \min(i_k - i_{k-1}, j_k - j_{k-1})$$

  - typ d)

    $$W(k) = \max(i_k - i_{k-1}, j_k - j_{k-1})$$

  kde $i_0 = j_0 = 0$.

  Pri použití určitých kombinácií lokálneho obmedzenia a váhovej funkcie môže vzniknúť situácia, kde niektoré úseky hľadanej ohýbacej cesty majú váhové ohodnotenie 0. Táto situácia nastáva napríklad pre kombináciu váhovej funkcie typu c) a lokálne obmedzenie znázornené na obrázku 3.5a. Lokálna vzdialenosť príslušných úsekov neprispieva k celkovej vzdialenosti, preto sa na riešenie takýchto situácií používa postup vyhladenia

[1] Pod transponovaným vektorom rozumiem vektor napísaný v stĺpci zhora nadol.

váh pozdĺž viacnásobného segmentu lokálnej cesty - obrázok 3.5b. Analogicky sa tento postup dá aplikovať aj na ďalšie lokálne obmedzenia.

a) Pred vyhladením: b) Po vyhladení:

Obrázok 3.5: Odstránenie nulových váh vyhladením

- **Normalizačný faktor:**

  Normalizačný faktor sa zavádza hlavne kvôli tomu, aby vyjadroval priemernú "chybu na krok" a umožňoval tak navzájom porovnávať chyby nerovnako dlhých postupností. Je to funkcia váhovej funkcie definovaná ako:

$$N(W) = \sum_{k=1}^{K} W(k)$$

  Normalizačný faktor bude potom rovný pre váhovú funkciu typu a)

$$N(W_a) = I + J$$

  pre váhovú funkciu typu b1)

$$N(W_{b1}) = I$$

  respektívne b2)

$$N(W_{b2}) = J$$

  Pre váhové funkcie typu c) a d) je ale hodnota normalizačného faktoru závislá na konkrétnom priebehu ohýbacej cesty. V takýchto prípadoch sa štandardne používa normalizačný faktor nezávislý na priebehu cesty. Napríklad:

$$N(W_c) = N(W_d) = I$$

3.2 Implementácia

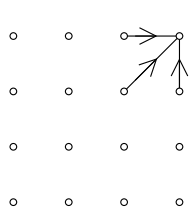
Keby sme chceli prechádzať všetky ohýbacie cesty a vybrať z nich tú najlepšiu, bola by časová náročnosť takéhoto algoritmu exponenciálna. Našťastie sa dá na riešenie tohoto problému efektívne aplikovať aj metóda dynamického programovania, pomocou ktorej sa časová náročnosť zníži na $O(I * J)$, kde I a J sú dĺžky porovnávaných postupností. Použijeme pomocnú maticu g, kde si budeme počítať doteraz najlepšiu naakumulovanú vzdialenosť od začiatku do daného bodu. Algoritmus bude potom vyzerať nasledovne:

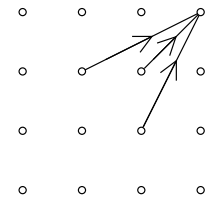
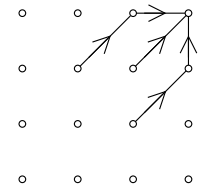
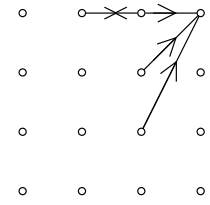
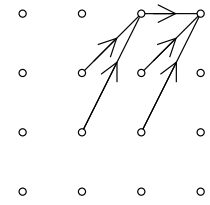
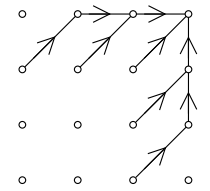
1. Inicializácia:
$$g(i_1, j_1) = d(i_1, j_1)W(1)$$
kde $d(i_1, j_1) = M(1, 1)$ je vzdialenosť začiatočných prvkov porovnávaných postupností a je to zároveň prvý prvok c_1 hľadanej ohýbacej cesty C.

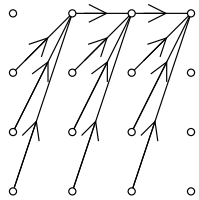
2. Rekurzia:
$$g(i_k, j_k) = \min\{g(i_{k-1}, j_{k-1}) + d(i_k, j_k)W(k)\}$$
kde $g(i_{k-1}, j_{k-1})$ je predošlý prvok z matice naakumulovaných vzdialeností, ktorý je presne určený použitým typom lokálneho obmedzenia.

3. Normalizácia:
$$DTW(A, B) = \frac{g(i_K, j_K)}{N(W)}$$
kde $i_K = I$ a $j_K = J$.

3.3 Používané kombinácie obmedzení a váh

Typ		α	β	$\mathbf{W(k)}$	$\mathbf{g(i,j)}$
1		0	∞	a	$\min \begin{cases} g(i, j-1) + d(i, j) \\ g(i-1, j-1) + 2d(i, j) \\ g(i-1, j) + d(i, j) \end{cases}$
				d	$\min \begin{cases} g(i, j-1) + d(i, j) \\ g(i-1, j-1) + d(i, j) \\ g(i-1, j) + d(i, j) \end{cases}$

Typ		α	β	$\mathbf{W}(\mathbf{k})$	$\mathbf{g}(\mathbf{i},\mathbf{j})$
2		$\frac{1}{2}$	2	a	$\min \begin{cases} g(i-1,j-2)+3d(i,j) \\ g(i-1,j-1)+2d(i,j) \\ g(i-2,j-1)+3d(i,j) \end{cases}$
				c	$\min \begin{cases} g(i-1,j-2)+d(i,j) \\ g(i-1,j-1)+d(i,j) \\ g(i-2,j-1)+d(i,j) \end{cases}$
3		$\frac{1}{2}$	2	a	$\min \begin{cases} g(i-1,j-2) \\ \quad +2d(i,j-1)+d(i,j) \\ g(i-1,j-1)+2d(i,j) \\ g(i-2,j-1) \\ \quad +2d(i-1,j)+d(i,j) \end{cases}$
4		$\frac{1}{2}$	2	b1	$\min \begin{cases} g(i-1,j)+xd(i,j) \\ g(i-1,j-1)+d(i,j) \\ g(i-1,j-2)+d(i,j) \end{cases}$ $x = 1$ pre $j_{k-1} \neq j_{k-2}$ $x = \infty$ pre $j_{k-1} = j_{k-2}$
5		$\frac{1}{2}$	2	d	$\min \begin{cases} g(i-1,j-2)+2d(i,j) \\ g(i-1,j-1)+d(i,j) \\ g(i-2,j-2) \\ \quad +2d(i-1,j)+d(i,j) \\ g(i-2,j-1) \\ \quad +d(i-1,j)+d(i,j) \end{cases}$
6		$\frac{1}{3}$	3	a	$\min \begin{cases} g(i-1,j-3)+2d(i,j-2) \\ \quad +d(i,j-1)+d(i,j) \\ g(i-1,j-2) \\ \quad +2d(i,j-1)+d(i,j) \\ g(i-1,j-1)+2d(i,j) \\ g(i-2,j-1) \\ \quad +2d(i-1,j)+d(i,j) \\ g(i-3,j-1)+2d(i-2,j) \\ \quad +d(i-1,j)+d(i,j) \end{cases}$

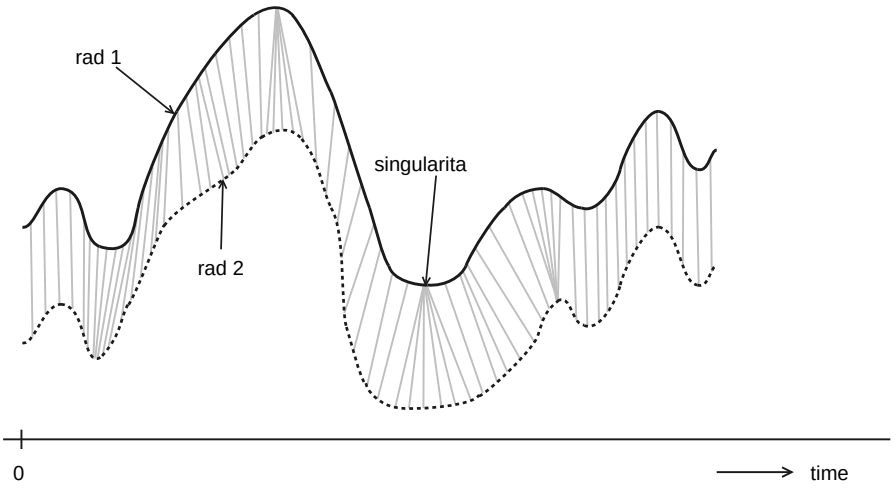
Typ		α	β	$\mathbf{W(k)}$	$\mathbf{g(i,j)}$
7		$\frac{1}{3}$	3	a	$\min \begin{cases} g(i-1,j-3)+4d(i,j) \\ g(i-1,j-2)+3d(i,j) \\ g(i-1,j-1)+2d(i,j) \\ g(i-2,j-3) \\ \quad +4d(i-1,j)+d(i,j) \\ g(i-2,j-2) \\ \quad +3d(i-1,j)+d(i,j) \\ g(i-2,j-1) \\ \quad +2d(i-1,j)+d(i,j) \\ g(i-3,j-3)+4d(i-2,j) \\ \quad +d(i-1,j)+d(i,j) \\ g(i-3,j-2)+3d(i-2,j) \\ \quad +d(i-1,j)+d(i,j) \\ g(i-3,j-1)+2d(i-2,j) \\ \quad +d(i-1,j)+d(i,j) \end{cases}$

3.4 Derivative Dynamic Time Warping - DDTW

Spolu s algoritmom DTW boli vyvíjané a testované aj rôzne jeho modifikácie a vylepšenia. Oproti klasickému DTW môžu dávať v určitých prípadoch lepšie výsledky. Jedným z takýchto vylepšení je aj Derivative Dynamic Time Warping. Tvorcovia tejto modifikácie si všimli, že klasický DTW algoritmus vytvára miesta postupností, kde jeden prvok jednej postupnosti je namapovaný na širokú oblasť prvkov druhej postupnosti. Takéto mapovanie jedného bodu sa volá singularita a môže vzniknúť, ak sú napríklad v postupnostiach časti, v ktorých sa hodnota signálu príliš nemení a jednému prvku takejto časti jednej postupnosti je namapovaný celý úsek druhej postupnosti. Lepšie si to predstaviť pre postupnosti jednorozmerných prvkov nám pomôže obrázok 3.6.

Jedná sa v podstate o korektné namapovanie, ale v skutočnosti si namapované body oboch radov nemusia zodpovedať. Ako riešenie tohoto problému bolo navrhnuté nebrať do mapovania jednotlivé prvky postupností, ale ich derivácie vypočítané pomocou vzťahu:

$$D(a_i) = \frac{(a_i - a_{i-1}) + \frac{a_{i+1} - a_{i-1}}{2}}{2}$$



Obrázok 3.6: Singularita

Prvé a posledné prvky postupností sa nepoužijú. Takáto jednoduchá modifikácia mala za následok zníženie výskytu singularít v mnohých oblastiach, kde klasické DTW singularity vytváralo. Podrobnejšie sa porovnaním DTW a DDTW zaoberá práca [11].

Kapitola 4

Návrh a implementácia

V tejto kapitole popisujem moju prácu na riešení zadaného problému. Popisujem tu moje nápady a pokusy, či už úspešné alebo nie, a popisujem aj k akým výsledkom viedli. Kapitolu som sa snažil členiť tak, aby jednotlivé myšlienky spolu súviseli, aj keď som sa nezamýšľal, neimplementoval a netestoval ich v poradí, v akom sú uvedené. Postupne po jednotlivých krokoch som popísal, ako som sa vysporiadal s jednotlivými čiastkovými problémami. Z ich riešení vyplynie výsledný algoritmus.

V nástroji, ktorý som vytvoril, je množstvo užívateľom nastaviteľných parametrov. V jednotlivých častiach spomínam, ktoré vlastnosti, s ktorými parametrami súvisia a aké zmeny by mohli vyvolať zmeny týchto parametrov. Používateľom môjho nástroja preto doporučujem prečítať si nasledujúcu kapitolu pre lepšie pochopenie, ako s ním úspešne pracovať.

4.1 Ako implementovať?

Ako prvý problém bolo treba rozhodnúť, ako implementovať moje riešenie zadaného problému. Nakoniec som si vybral programovací jazyk java. Toto rozhodnutie som urobil z viacerých dôvodov:

- Nezávislosť na platforme - Chcel som aby mnou vytvorený nástroj bol dobre použiteľný na rôznych operačných systémoch
- Veľa hotových vecí - O jave je známe, že keď v nej ideme niečo nové vytvoriť je rozumné sa poobzerať, či to už náhodou vytvorené nebolo. Vo veľa prípadoch stačí chvíľka hľadania na internete a človek si ušetrí množstvo času a programovania.
- Offline riešenie - Programy napísané v jave sú vo všeobecnosti pomalšie, ako rovnaké programy v iných programovacích jazykoch. Riešenie môjho problému ale nemusí prebiehať v reálnom čase, ako pri väčšine aplikácií riešiacich rôzne problémy z rozpoznávania reči.
- Vzdelávanie - Jedným z dôvodov prečo som si vybral programovací jazyk java, bol aj môj záujem rozšíriť svoje vedomosti v tejto oblasti.

4.2 Vstupy a výstupy

Na začiatok je dobré spomenúť, čo je na vstupe, a aký je očakávaný výstup. V tomto prípade primárne vstupy pre výpočet budú:

- *suvisle.mp3* - súvisle nahovorený príbeh vo formáte mp3 (prípustný je aj formát wav).
- *izolovane.mp3* - izolovane (po jednotlivých slovách s prestávkami medzi nimi) nahovorený ten istý príbeh vo formáte mp3 (alebo wav)
- *izolovane.mlab* - súbor obsahujúci časy začiatkov a koncov izolovane nahovorených slov. Súbor používa htk formát [12] - v riadkoch sa striedajú slová s pauzami a každý riadok obsahuje tri hodnoty oddelené medzerami:
  - začiatok slova (pauzy) - Celočíselná hodnota, ktorej prenásobením 10^{-7} dostaneme čas v sekundách.

- koniec slova (pauzy) - Celočíselná hodnota, ktorej prenásobením 10^{-7} dostaneme čas v sekundách.
- *slovo* alebo "*#sil*" (ak riadok reprezentuje pauzu)

Súbor môže vyzerať napríklad takto:

0 1500000 #sil
1500000 10800000 slovo_1
10800000 15234437 #sil
15234437 26400000 slovo_2
26400000 33200000 #sil
33200000 36767729 slovo_3
36767729 44000000 #sil
⋮

Ako primárny výstup algoritmus vytvorí súbor *suvisle.mlab* - súbor obsahujúci začiatky a konce súvisle nahovorených slov podobného formátu ako súbor *izolovane.mlab*, s tým rozdielom, že časy už nie sú celočíselné. Sú uvedené priamo v sekundách a súbor už neobsahuje riadky s pauzami. Súbor môže vyzerať napríklad takto:

0,279	0,726	slovo_1
0,801	1,115	slovo_2
1,150	1,202	slovo_3
⋮		

Okrem primárnych vstupov a výstupov mnou implementovaný nástroj spracuje a vytvára aj niekoľko sekundárnych vstupov a výstupov. Ako nepovinný vstup sa môže uviesť (ako parameter pri spustení) úplný názov konfiguračného súboru, ktorý je na začiatku behu algoritmu načítaný a údaje v ňom uvedené sa použijú na nastavenie parametrov algoritmu. Ak sa konfiguračný súbor neuvedie, alebo neobsahuje nastavenia všetkých parametrov, tak sa pre všetky, alebo pre neuvedené parametre použijú defaultné nastavenia. Konfiguračný súbor musí mať presne určený tvar, inak bude vypísané varovné hlásenie a použijú sa defaultné nastavenia. Každý riadok konfiguračného súboru zodpovedá nastaveniu jedného parametra. Riadok začína názvom parametra (veľkými písmenami), potom nasleduje minimálne jedna medzera, znak "=", minimálne

jedna medzera a hodnota parametra (podľa typu parametra - celé číslo, číslo s pohyblivou rádovou čiarkou alebo reťazec). Jednotlivé parametre môžu mať ľubovolné poradie a prázdne riadky sú tiež povolené. Príklad konfiguračného súboru:

```
DLZKA_OKIENKA = 512
VAHA_CHYBY = 0.005
VYSTUP = suvisle.mlab
⋮
```

Pre primárne vstupy je dôležité nastavenie cesty pracovného adresára. Nastavíme ju pomocou parametra

CESTA

Defaultná hodnota cesty je prázdny reťazec (pracuje sa v adresári, z ktorého spúšťam samotný program). Parameter

SUVISLE

určuje názov súboru súvisle nahovoreného príbehu (vo formáte mp3 alebo wav) nachádzajúceho sa v pracovnom adresári. Defaultná hodnota je *suvisle.mp3*. Podobne parameter

IZOLOVANE

určuje názov súboru izolovane nahovoreného príbehu (vo formáte mp3 alebo wav) nachádzajúceho sa v pracovnom adresári. Defaultná hodnota je *izolovane.mp3*. Časy začiatkov a koncov izolovaných slov sa načítajú zo súboru určeným parametrom

IZOLOVANE_INDEXY

s defaultnou hodnotou *izolovane.mlab* a názov súboru, do ktorého sa uložia výstupné časy začiatkov a koncov súvisle nahovorených slov je určený parametrom

SUVISLE_INDEXY

Defaultná hodnota je *suvisle.mlab*.

Ostatné sekundárne vstupy a výstupy spomeniem a vysvetlím v príslušných častiach tejto kapitoly.

4.3 Spracovanie vstupov

Na začiatku výpočtu je potrebné spracovať vstupy do vhodnej podoby, aby sa s nimi dalo ďalej narábať. Najprv je potrebné zmeniť súbory formátu mp3 na súbory formátu wav. Na to som použil nástroj - *MP3SPI*[1]. Jedná sa o open source projekt - plugin pre J2SE[2]. Zo získaného wav súboru, ak je viackanálový, sa zoberie len jeden kanál a vyextrahujú sa 16-bitové celočíselné hodnoty intenzít signálu a vzorkovacia frekvencia (sample rate).

4.3.1 Stanovenie hraníc blokov a slov a ich postupné dolaďovanie

Vo výpočte sa nepracuje s celým signálom naraz. To by bolo neúnosne náročné. Na začiatku treba signál rozdeliť (určiť hranice) na vhodne dlhé úseky a následne pracovať s nimi.

Izolovane nahovorený príbeh sa jednoducho rozdelí na jednotlivé slová podľa časov ich začiatkov a koncov, ktoré sú uvedené vo vstupnom súbore *izolovane.mlab*. Keďže však tento súbor neobsahuje hranice slov určené tesne (na začiatku a na konci sa nachádza väčšinou kúsok "ticha"), bolo by vhodné ich určiť tesnejšie, nakoľko pri ich súvislom vyslovovaní sa postupné zvýšenie intenzity signálu na začiatku slova a jej postupné zníženie na konci slova väčšinou stratia. Pomocou intenzity signálu som skúsil určiť tesné ohraničenia pre jednotlivé izolovane nahovorené slová. Prinieslo to zlepšenie, ale nie dostatočné.

Slová v súvisle prečítanom texte zodpovedajú tesne ohraničeným izolovaným slovám vtedy, keď sú vyslovované okamžite za sebou. Ak súvislý text obsahuje pauzu (napríklad na nádych), tak príslušné slová okolo nej si zachovávajú postupné zníženie (zvýšenie) intenzity signálu. Preto by bolo vhodné, aby sa použilo tesné ohraničenie slova iba vtedy, keď sa to bude hodiť. Riešenie tohto problému popisujem až neskôr pri samotnom algoritme, ale je potrebné si zapamätať pre každé izolované slovo, ako by vyzeralo jeho tesné ohraničenie. To sa uskutoční v tejto fáze.

Súvisle prečítaný príbeh som sa rozhodol rozdeliť na bloky. Toto rozdelenie funguje na princípe počítania intenzity signálu. Pri jej poklese (prestávka na nádych) pod určitú hladinu, určím koniec bloku a následne pri jej opä-

[1] Service Provider Interface
[2] Java 2 Standard Edition

tovnom zvýšení, určím začiatok ďalšieho bloku. Vo väčšine prípadov takto dostanem bloky, ktorých dĺžka nepresahuje 5 sekúnd. Ak by som dostal výrazne dlhší blok, mohlo by to znamenať výrazné zvýšenie času potrebného na beh algoritmu. Preto je určená maximálna prípustná dĺžka bloku, a ak dostanem nejaký blok dlhší ako je táto hranica, tak sa ho algoritmus pokúša rozdeliť na menšie zvýšením hraničnej hladiny intenzity, alebo skrátením časového intervalu, z akého je intenzita počítaná.

Následne sú určené tesné ohraničenia jednotlivých blokov podobne ako to bolo pri izolovaných slovách, nie však až tak tesné.

Pri rozdeľovaní signálu na bloky (slová) a pri určovaní ich tesnejších ohraničení vystupuje množstvo parametrov hraničných hladín intenzít a dĺžok intervalov. Snažil som sa nájsť ideálne nastavenia, ale keďže predpokladám, že používateľ môže skúšať spracovať aj výrazne neideálne vstupy, dal som možnosť ich prenastavenia pomocou konfiguračného súboru. Parameter

HRANICNA_INTENZITA

slúži na rozdelenie súvisle prečítaného textu do blokov. Určuje hranicu intenzity zvuku, nad ktorou sa signál vyhodnotí ako slovo a pod ňou ako ticho. Jej defaultná hodnota je *400*. Pri hlasnejších nahrávkach s množstvom šumu sa môže táto hranica zvýšiť, pri tichších znížiť.

Ako som už spomenul, pri určovaní hraníc blokov je dôležité, aby bloky neboli príliš dlhé a rovnako by nemali byť ani príliš krátke. To zabezpečujú dva parametre. Prvý určuje maximálnu povolenú dĺžku bloku

MAXIMALNA_POVOLENA_DLZKA_BLOKU

Ak je dĺžka väčšia, algoritmus skúsi daný blok rozdeliť znova s posunutými hranicami. Defaultná hodnota je *8.0* sekúnd. Druhý parameter určuje minimálnu povolenú dĺžku bloku

MINIMALNA_POVOLENA_DLZKA_BLOKU

Bloky kratšie ako táto hodnota sa neberú do úvahy. Defaultná hodnota je *0.45* sekundy.

Pri dolaďovaní hraníc ešte treba určiť, ako tesne sa má toto doladenie uskutočniť. Pokusne som zistil, že lepšie, ako stanoviť hraničnú intenzitu absolútne, je vychádzať z priemernej intenzity práve dolaďovaného úseku signálu (blok alebo slovo). Pri izolovaných slovách to môžeme nastaviť pomocou parametra

OSEKANIE_IZOLOVANYCH

Jeho hodnota nám hovorí od koľko-násobku priemernej intenzity signálu slova sa majú označiť doladené (tesnejšie) hranice daného slova. Jeho defaultná hodnota je *1*. Pri tesnom ohraničovaní blokov použijeme parameter

OSEKANIE_BLOKOV

ktorý má analogický význam, ibaže tu sa používa nižšia hodnota (v opačnom prípade môže dochádzať k strate signálu - tichých slov na začiatku a na konci bloku). Defaultná hodnota je *0.25*

Pod pojmom intenzita myslím priemernú intenzitu počítanú z určitého intervalu. Ak je tento interval príliš dlhý, mohlo by sa stať, že prídeme o nejaké krátke slovo (prípadne budú vznikať príliš dlhé bloky, keď neidentifikujem pauzu), ak je naopak príliš krátky, dochádza k rozdeľovaniu blokov v strede dlhších slov s krátkou prestávkou vo vnútri (prípadne vzniká množstvo krátkych blokov, ktoré neobsahujú žiadne slovo, iba sa na tom mieste trochu výraznejšie zvýšila intenzita šumu v pozadí). Dĺžka tohto intervalu sa dá nastaviť pre rozdeľovanie signálu do blokov pomocou parametra

DLZKA_INTERVALU_INTENZITY_ROZSEKAVANIE

Jeho defaultnú hodnotu som pokusne určil na *0.3* sekundy. Pri doladťovaní (blokov aj slov) môže byť dĺžka intervalu kratšia. Dá sa nastaviť pomocou parametra

DLZKA_INTERVALU_INTENZITY_OSEKAVANIE

a jej defaultná hodnota je *0.05* sekundy.

Ak pri doladťovaní hraníc dosiahnem kratšie slovo ako určitý prípustný limit, doladťovanie sa opakuje s mierne zmenenými príslušnými parametrami, kým sa daná hranica nedosiahne. Táto hranica je určená parametrom

MIN_DLAZKA_IZOLOVANYCH

a jej defaultná hodnota je *0.1* sekundy.

Všetky defaultné hodnoty tu uvedené som určil pokusmi a pozorovaním správania sa algoritmu určovania hraníc a ich doladťovania na viacerých vstupoch.

4.3.2 Výpočet vektorov príznakov

Následne z takto upravených vstupov vypočítame ich MFCC koeficienty. Ako som už spomínal v prehľadovej kapitole, už množstvo prác z rozpoznávania reči sa zaoberalo témou porovnávania výsledkov dosiahnutých pomocou reprezentácie signálu rôznymi vektormi príznakov. Vo všetkých prácach, s ktorými som sa stretol, vždy vyšli najlepšie výsledky pre MFCC koeficienty. Preto som sa rozhodol použiť ich bez ďalšieho testovania a porovnávania výsledkov aj pre iné koeficienty. Na ich výpočet som použil *jAudioFeatureExtractor* - java balíček umožňujúci spracovanie a extrakciu rôznych typov príznakov zo zvukového signálu. Autori ho použili vo svojom projekte jAudio [13] - nástroji na extrakciu príznakov s jednoduchým GUI rozhraním alebo príkazovým riadkom. Cieľom projektu bolo vytvoriť zjednotenie všetkých algoritmov a postupov používaných pri spracovaní signálu s možnosťou pridávania nových metód. Môj algoritmus využíva len malú časť schopností spomenutého java balíčku.

Výpočet MFCC je bližšie popísaný v časti 2.2.5. Po skončení výpočtu dostaneme postupnosť 13-rozmerných vektorov príznakov.

Moja aplikácia umožňuje okrem výpočtu samotných MFCC vektorov vypočítať a ďalej pracovať aj s 26 rozmernými vektormi, ktorých prvú polovicu tvoria pôvodné MFCC vektory a druhú delta koeficienty, ktoré sú počítané podľa predpisu spomínanom pri DDTW v časti 3.4. Rovnako je možné pracovať aj so samostatnými delta koeficientami. Na uplatnenie nášho rozhodnutia slúži parameter

DELTY

ktorý nadobúda nasledovné hodnoty:

- *0* - vypočítajú sa len MFCC
- *1* - vypočítajú sa aj MFCC, aj delta koeficienty
- *2* - vypočítajú sa len delta koeficienty

Jeho defaultná hodnota je *0*.

Podľa mojich pozorovaní, a tiež, po zamyslení sa nad tým, aký problém sa snaží riešiť DDTW, som dospel k tomu, že použitie delta koeficientov na môj

problém neprináša nič lepšie, ako použitie samotných MFCC. Pri použití kombinovaných 26-rozmerných vektorov som dosahoval porovnateľné výsledky, ale prinieslo to zhoršenie časovej náročnosti. Použitie samotných delta koeficientov síce zhoršenie časovej náročnosti neprinieslo, ale výsledky to tiež nezlepšilo.

4.4 Použitý typ DTW a jeho modifikácie

Ako základ pre svoj algoritmus som sa rozhodol použiť algoritmus DTW s najčastejšie používaným spôsobom lokálnych obmedzení. Je to prvý typ DTW z tabuľky používaných obmedzení a váh (3.3) z predchádzajúcej kapitoly. Skúšal som aj iné typy DTW, ale lepšie výsledky som nedosahoval a implementácia bola komplikovanejšia. Pre zvolený typ som navrhol a testoval viacero jeho vylepšení a modifikácií, ktoré som potom zahrnul do výsledného algoritmu.

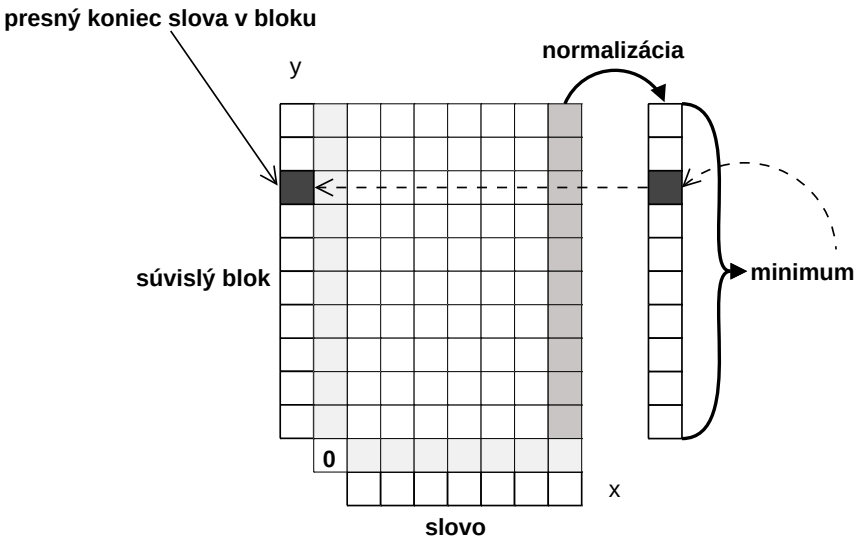
4.4.1 Slajdovacie DTW

Jedným z prvých vylepšení, ktoré som vyskúšal, bola modifikácia DTW, ktorá umožňuje hľadať presný koniec (začiatok) namapovania jednej postupnosti do druhej. Predpokladajme, že namapovávaný úsek signálu (izolované slovo) pri behu DTW umiestnim na os x (vodorovná os) a signál, v ktorom sa snažím lokalizovať polohu tohto slova (blok súvisle prečítaného textu) umiestnim na os y (zvislá os). Celé to potom funguje na jednoduchom princípe:

1. Urob klasický výpočet DTW na daných postupnostiach.
2. Normalizuj posledný (pravý) stĺpec tabuľky čiastkových vzdialeností g
3. Postupuj zhora nadol (celým, alebo len po určitú jeho časť) normalizovaným posledným stĺpcom tabuľky g a nájdi minimum. Tam bude ležať koniec namapovávaného slova.
4. Takto určený koniec označ ako "nový začiatok" a analogickým postupom pre reverznuté postupnosti nájdi presný začiatok slova.

Samozrejme sa to nedá použiť dobre, ak sa snažíme nájsť polohu slova v dlhej postupnosti súvisle prečítaných slov. Fungovať by to malo hlavne v prípadoch, keď máme krátky úsek signálu, vieme, že slovo by sa v ňom malo

nachádzať a chcem určiť jeho presný začiatok a koniec. Prvé tri body tohto algoritmu sú znázornené na obrázku 4.1.



Obrázok 4.1: Slajdovacie DTW

Táto modifikácia má aj svoje nedostatky. Napríklad, ak je na konci slova dlhá samohláska, môže sa stať, že v súvislom bloku sa namapuje len jej začiatok a podobne.

4.4.2 Preskakovacie DTW

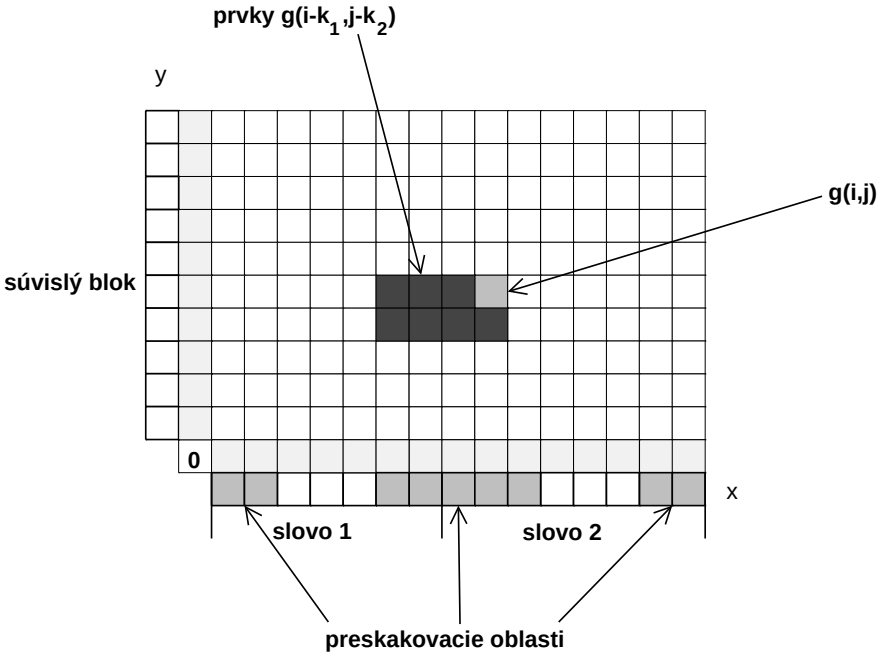
Túto modifikáciu som vymyslel, keď som sa zamýšľal nad tým, ako vyriešiť už spomínaný problém dolaďovania hraníc izolovaných slov tak, aby sa niekedy bralo tesné ohraničenie a inokedy nie. Predpokladajme, že chceme namapovať skupinu izolovaných slov zoradených za sebou (umiestnených na osi x) na súvisle nahovorenú postupnosť týchto slov (umiestnených na osi y). Pre každé izolované slovo budeme mať vyznačené oblasti začiatkov a koncov (preskakovacie oblasti). Chceme dosiahnuť, aby tieto oblasti (celé alebo ich časti) mohli byť vynechané, ak by som tak dosiahlo lepšie namapovanie. Modifikácia vyzerá nasledovne. Pri behu zvoleného typu DTW (typ 1) mimo týchto oblastí fungujem podľa štandardného algoritmu:

$$g(i,j) = \min\{g(i,j-1) + d(i,j), g(i-1,j-1) + d(i,j), g(i-1,j) + d(i,j)\}$$

kde g je matica čiastkových vzdialeností a $d(x, y)$ je použitý typ vzdialenosti vektorov x a y. Keď ale vypĺňam maticu g na miestach preskakovacích oblastí, chcem mať možnosť preskočiť ľubovoľne dlhý úsek danej oblasti. To dosiahnem nasledovnou úpravou:

$$g(i, j) = \min\{g(i - k_1, j - k_2) + d(i, j)\}$$

kde $k_1 = 0, \ldots, n$, tak aby všetky $i - k_1$ ležali v danej oblasti a $k_2 = 0, 1$. Pre dve slová som to znázornil na obrázku 4.2.



Obrázok 4.2: Preskakovacie DTW

4.4.3 Počítanie vzdialeností

MFCC vektory sú charakteristické tým, že jednotlivé súradnice sa rádovo od seba odlišujú (akoby mali inú mierku). Z pozorovaní vyplynulo, že MFCC zachytávajúce nižšie frekvencie majú vyššie hodnoty ako MFCC zachytávajúce vyššie frekvencie. Toto by mohlo spôsobiť, že pri počítaní vzdialenosti dvoch MFCC vektorov sa koeficienty nižších rádov (nižšie frekvencie) budú podieľať na celkovej vzdialenosti výrazne vyššou mierou, ako koeficienty vyšších rádov (vyššie frekvencie). Stojí za uváženie, či je to dobre alebo zle.

Ja som sa snažil tieto rozdiely medzi jednotlivými koeficientami v rámci vektora najprv vyrovnávať pomocou funkcie *lifter*, o ktorej som sa dočítal v [12]. Idea je taká, že sa každý prvok x_i MFCC vektora x upraví pomocou vzťahu:

$$x'_i = (1 + \frac{L}{2} \sin \frac{\pi i}{L}) x_i$$

kde x_i je i-ta súradnica vektora x, x'_i je nová i-ta súradnica vektora x a L je kladné celé číslo, ktorého veľkosť určuje, ako veľmi sa zvýšia hodnoty MFCC koeficientov. Následne som na výpočet vzdialeností dvoch vektorov x a y použil vzťah:

$$d(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2$$

kde n je dimenzia vektorov. Tento spôsob pri vhodne zvolenom L prináša približné vyrovnanie jednotlivých súradníc MFCC vektorov, ale je to len umelý spôsob, ktorý neberie do úvahy, ako presne vektory vyzerajú a rádovo aké veľké rozdiely medzi súradnicami skutočne sú.

Skúsil som spočítať zo všetkých MFCC vektorov (aj súvisle aj izolovane nahovoreného príbehu) strednú hodnotu pre každú súradnicu a rovnako rozptyl (disperziu) každej súradnice, počítaný nezávisle zo všetkých vektorov. Potom som na počítanie vzdialenosti dvoch vektorov x a y použil nasledovný vzťah:

$$d(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2 \frac{1}{D_i}$$

kde D_i je disperzia i-tej súradnice MFCC vektorov pre dané vstupy. Takto upravené počítanie vzdialenosti prinieslo výrazné zlepšenie oproti použitiu funkcie *lifter*, pretože priamo zo vstupných dát vypočítavalo, aké veľké hodnoty nadobúdajú v priemere príslušné súradnice a podľa toho im bola pri výpočte pridelená váha, akou sa budú podieľať na celkovej vzdialenosti.

Následne som skúsil nepočítať disperzie nezávisle, ale so vstupných dát vytvoriť strednú hodnotu a kovariačnú maticu upravenú na diagonálny tvar, čím získame viacrozmerný gausián. Disperzie jednotlivých súradníc potom ležia na diagonále kovaričnej matice. Tento gausián vlastne charakterizuje zvukový prejav daného rečníka. Tomu prispôsobíme jednotlivé váhy MFCC koeficientov.

MFCC vektory nemusia byť vo svojom vektorovom priestore rovnomerne rozložené. Spravidla vytvárajú akési zhluky. Preto by bolo možno vhodné, nemať len jeden gausián, ale viacero. Intuitívne si za tým môžeme predstaviť, že

keď mám gausiány pre daného rečníka napríklad 2, tak MFCC vektory prislúchajúce tichu (jeden zhluk) sa napríklad budú porovnávať s inými váhami ako MFCC vektory prislúchajúce reči (druhý zhluk). Mohli by sme tak dosiahnuť lepšie výsledky. Na vytvorenie viacerých gausiánov som použil na to určený známy algoritmus:

1. Spočítaj strednú hodnotu zo všetkých MFCC vektorov. Výsledkom je priemerný vektor μ_0.

2. Spočítaj kovariačnú maticu Σ'_0 podľa vzťahu:

$$\Sigma'_0 = \frac{\sum_{i=1}^{K}(v_i - \mu_0)^T(v_i - \mu_0)}{K}$$

   kde K je počet všetkých MFCC vektorov vstupu, v_i je i-ty MFCC vektor a T značí transponovaný vektor[3].

3. Takto získaná matica je symetrická a ortogonálnymi úpravami (naraz vykonávam úpravy aj na riadku aj na príslušnom stĺpci) z nej urob diagonálnu maticu Σ_0. Hodnoty μ_0 a Σ_0 určujú prvý n-rozmerný gausián G_0 (kde n je dimenzia MFCC vektorov). Hodnotu b_0 takéhoto gausiánu vo vektore v môžeme vypočítať pomocou vzťahu:

$$b_0(v) = \frac{1}{\sqrt{(2\pi)^n|\Sigma_0|}} e^{-\frac{1}{2}(v-\mu_0)(\Sigma_0)^{-1}(v-\mu_0)^T}$$

   kde $|\Sigma_0|$ je determinant matice Σ_0 a $(\Sigma_0)^{-1}$ je ku nej inverzná matica.

Ak už máme určitý počet gausiánov p a chcem ho zvýšiť na $p+1$ postupujeme nasledovne:

1. Z gausiánov, ktoré už máš nájdi ten najširší gausián G_i (Vezmi ten, ktorého matica Σ_i, bude mať na diagonále najväčšie hodnoty).

2. Skopíruj diagonálnu kovariačnú maticu Σ_i aj strednú hodnotu μ_i 2-krát, pričom v jednom prípade pripočítaj k strednej hodnote malú kladnú konštantu ε a v druhom prípade ju odpočítaj. Dostaneš dve nové dvojice strednej hodnoty a kovariačnej matice určujúce dva nové gausiány G'_{i_1} a G'_{i_2}:

$$G'_{i_1}: \qquad \mu'_{i_1} = \mu_i + \varepsilon \qquad \Sigma'_{i_1} = \Sigma_i$$
$$G'_{i_2}: \qquad \mu'_{i_2} = \mu_i - \varepsilon \qquad \Sigma'_{i_2} = \Sigma_i$$

[3]Pod transponovaným vektorom rozumiem vektor napísaný v stĺpci zhora nadol.

3. Obidve stredné hodnoty μ'_{i_1} a μ'_{i_2} zmeň na nové stredné hodnoty μ_{i_1} a μ_{i_2} podľa vzťahu:

$$\mu_{i_x} = \frac{\sum_{i=1}^{K} v_i b'_{i_x}(v_i)}{\sum_{i=1}^{K} b'_{i_x}(v_i)}$$

kde $x = 1, 2$ a $b'_{i_x}(v_i)$ je hodnota gausiánu G'_{i_x} vo vektore v_i. Dostaneš dva nové gausiány G''_{i_1} a G''_{i_2} určené dvojicami:

$$G''_{i_1} : \qquad \mu_{i_1} \qquad \Sigma'_{i_1}$$

$$G''_{i_2} : \qquad \mu_{i_2} \qquad \Sigma'_{i_2}$$

4. Následne obidve kovariačné matice Σ'_{i_1} a Σ'_{i_2} analogicky zmeň na nové kovariačné matice Σ_{i_1} a Σ_{i_2} podľa vzťahu:

$$\Sigma_{i_x} = \frac{\sum_{i=1}^{K} v_i^T v_i b''_{i_x}(v_i)}{\sum_{i=1}^{K} b''_{i_x}(v_i)}$$

Dostaneš dva nové gausiány G_{i_1} a G_{i_2} určené dvojicami:

$$G_{i_1} : \qquad \mu_{i_1} \qquad \Sigma_{i_1}$$

$$G_{i_2} : \qquad \mu_{i_2} \qquad \Sigma_{i_2}$$

Týmito gausiánmi nahraď pôvodný gausián G_i, a ak potrebuješ viac gausiánov pokračuj znova krokom 1.

Počítanie vzdialeností dvoch MFCC vektorov x a y pomocou viacerých gausiánov sa potom uskutočňuje tak, že vypočítavam pre každý gausián súčin ich hodnôt v oboch vektoroch:

$$b_i(x)b_i(y)$$

kde $i = 1, \ldots, P$ a P je počet gausiánov. Nájdem ten gausián, pri ktorom je tento súčin najväčší (obidva vektory ležia najbližšie k jeho stredu), a ako váhy jednotlivých súradníc použijem prevrátené hodnoty diagonálnych prvkov δ_{ii} kovariačnej matice víťazného gausiánu (disperzie):

$$d(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2 \frac{1}{\delta_{ii}}$$

Pri behu algoritmu, kde počítanie vzdialenosti dvoch vektorov je najčastejšie sa vyskytujúca operácia, počítanie hodnôt gausiánov trvalo neakceptovateľne dlho. Časová náročnosť algoritmu sa mnohonásobne zvýšila. Tento problém

som pri implementácii vyriešil predvypočítaním hodnôt gausiánov pre všetky vstupné MFCC vektory.

Na nastavenie, ako sa majú vzdialenosti počítať, slúži v konfiguračnom súbore parameter

POCITANIE_VZDIALENOSTI

nadobúdajúci nasledovné hodnoty:

- *-1* - Použijú sa pôvodné MFCC koeficienty bez akejkoľvek úpravy s rovnakou váhou
- *0* - Zo vstupov sa vypočíta (nezávisle) disperzia každej súradnice a váha jednotlivých súradníc je potom jej prevrátená hodnota
- *n* ,kde *n* je kladné celé číslo väčšie ako 0 - vyššie spomínaným algoritmom sa vypočíta *n* gausiánov a váhovanie pri výpočte prebieha tak, ako bolo spomínané vyššie

Defaultná hodnota je *0*.

Ak je hodnota predchádzajúceho parametra väčšia ako 1, máme možnosť pomocou parametra

POSUN_STREDOV_GAUSIANOV

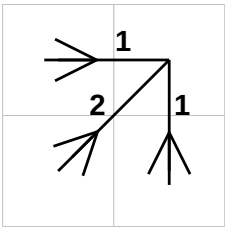
nastaviť hodnotu ε (ako veľmi sa budú posúvať stredy nových dvoch gausiánov oproti stredu pôvodného gausiánu z ktorého vznikli). Hodnota určuje o koľko ($*100\%$ smerodajnej odchýlky príslušnej súradnice) sa stredy posunú jedným alebo druhým smerom. Defaultná hodnota je *0.25*.

Pri mojich testoch som dosahoval veľmi dobré výsledky pri nezávislom počítaní jednotlivých disperzií. Keď som použil jeden gausián výsledky sa nepatrne zhoršili, ale použitím viacerých (dvoch, troch, ...) sa znova zlepšili.

Použitie váhovania sa ukázalo vhodné hlavne pri pekne nahovorených príbehoch s nízkou intenzitou šumu v pozadí. U takýchto vstupov to prinieslo zlepšenie. Keď ale máme intenzitu šumu v pozadí dosť vysokú, ukázalo sa, že je lepšie nepoužiť váhovanie. Tým pádom budú mať pri mapovaní väčšiu váhu koeficienty nižších frekvencií (vyplýva to z hodnôt aké nadobúdajú jednotlivé súradnice MFCC vektorov). To spôsobí, že slová už nebudú namapované až s takou presnosťou (vysokofrekvenčný šum a zvuky frikatív sú pri mapovaní akoby odsunuté do úzadia), ale v určitých prípadoch to môže priniesť výrazne lepšie výsledky.

4.4.4 Normovanie

Pri použitých lokálnych obmedzeniach algoritmu DTW sa štandardne používajú dva typy váhových funkcií a k nim prislúchajúce dva typy normalizačných faktorov. Buď je váha na ohýbacej ceste vo vodorovnom a zvislom smere 1 a v diagonálnom smere 2. (Obrázok 4.3) Použitý normalizačný faktor je v takomto prípade $I + J$ (súčet dĺžok mapovaných postupností). Alebo je váha vo všet-



Obrázok 4.3: Štandardne používaná váhovacia funkcia

kých smeroch 1 a ako normalizačný faktor sa štandardne používa dĺžka z jednej napasovávaných postupností. Mne sa na prvom prípade nepáčilo, že dané váhovanie mi vlastne hovorí: "Keď ohýbacia cesta smeruje najprv vodorovne a následne zvisle, je to rovnako dobré, akoby smerovala raz diagonálne" Podľa mňa by pohyb po diagonále nemal byť znevýhodnený. To sa deje v druhom prípade, ale v ňom používaná norma (keďže je určená nezávisle na priebehu ohýbacej cesty) môže výsledky dosť skresľovať. Preto som sa rozhodol používať váhovanie ako v druhom prípade a ako normu použiť skutočnú dĺžku ohýbacej cesty, ktorú zisťujem spätným prechodom po jej nájdení. To mi síce za normálnych okolností zhoršuje časovú náročnosť, ale v časovo kľúčových fázach behu algoritmu sa spätný prechod aj tak uskutočňuje kvôli nájdeniu začiatkov a koncov slov v bloku, takže to nespôsobí skoro žiadne zhoršenie časovej zložitosti a zdá sa, že výsledky by mohli byť priateľnejšie.

Takéto váhovanie a norma funguje rovnako aj pri použití preskakovacieho DTW. Jednoducho sa preskočený úsek váhuje jednotkou a aj do dĺžky ohýbacej cesty pridá len jednotku.

4.5 Vývoj algoritmu

Postupne, ako som implementoval jednotlivé vylepšenia, výsledný algoritmus zaznamenal tri kľúčové štádiá:

4.5.1 Algoritmus č. 1

Prvá myšlienka, ako by to mohlo celé fungovať, bola nasledovná:

1. Vezmi ďalšie izolované slovo (postupnosť MFCC vektorov prislúchajúca ďalšiemu izolovanému slovu) s_k v poradí.

2. Na základe pomeru jeho dĺžky k celkovej dĺžke všetkých nasledujúcich izolovaných slov (+ jeho dĺžka) odhadni, aká veľká časť ešte nepoužitého signálu (postupnosti MFCC vektorov) súvisle nahovoreného textu by mu mala asi zodpovedať. (Mala by byť v rovnakom pomere k zostávajúcej dĺžke signálu súvisle nahovoreného textu)

   $$l(b_k) = \frac{l(s_k)}{\sum_{i=k}^{N} l(s_i)} l(B_{k...N})$$

   kde l značí dĺžku, s_k je k-te izolované slovo v poradí, b_k je k nemu hľadaný blok signálu súvisle nahovoreného textu, $B_{k...N}$ je zostávajúci signál súvisle nahovoreného textu (začínajúci blokom b_k) a N je celkový počet slov.

3. Pomocou slajdovacieho DTW skús namapovať toto izolované slovo na danú odhadnutú časť signálu súvisle nahovoreného textu s nejakou toleranciou chyby odhadu.

4. Následne vezmi ďalšie izolované slovo a pokračuj bodom 1.

Tento algoritmus bol dosť neúspešný. Veľmi ľahko vznikali v riešení chyby (namapovovanie s určitým posunom oproti správnemu riešeniu). A vytvorená chyba sa šírila ďalej s minimálnou šancou na opravu.

4.5.2 Algoritmus č. 2

Následne som vyskúšal využiť možnosť rozdelenia signálu súvisle nahovoreného príbehu pomocou intenzity signálu na bloky a skúšať mapovať tieto bloky namiesto samostatných slov. Korektným rozdelením na bloky získam

novú pomôcku v tom, že viem, že jedno slovo musí mať začiatok aj koniec v tom istom bloku. Algoritmus vyzerá nasledovne (predpokladám, že už mám signál súvisle nahovoreného textu rozdelený na bloky postupom spomínaným vyššie v tejto kapitole, a tiež, že už mám vyznačené tesné ohraničenia pre všetky izolované slová):

1. Vezmi nasledujúci blok b_k signálu súvisle nahovoreného textu.
2. Na základe pomerov dĺžok (analogicky ako v algoritme č. 1) odhadni počet q nasledujúcich izolovaných slov prislúchajúcich danému bloku b_k.
3. Usporiadaj si tieto izolované slová za sebou a použitím preskakovacieho DTW ich skús namapovať do pripraveného bloku b_k.
4. Toto namapovanie nevyskúšaj len pre odhadnutý počet slov, ale aj pre iné počty slov ($q + r$ kde $r = -R, \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots, R$) a ako výsledok zober to namapovanie, pri ktorom je chyba DTW najmenšia.
5. Vráť sa na krok 1. a rovnakým spôsobom pokračuj s nasledujúcim blokom b_{k+1}, až kým nemáš namapované všetky bloky.
6. Na konci pre každé slovo v rámci jemu prideleného úseku v príslušnom bloku, nájdi jeho presný začiatok a koniec pomocou slajdovacieho DTW.

Tento algoritmus zaznamenal výrazne väčší úspech oproti predošlému. Keď vznikla v riešení chyba, postupným skúšaním namapovania rôzneho počtu slov na každý blok, sa nešírila až do konca, ale časom sa napravila a nasledovné bloky boli znova určované správne. Chybou je, že tento čas nápravy je spravidla pomerne dlhý. Tento problém som nakoniec s istým úspechom vyriešil v algoritme č. 3.

4.5.3 Algoritmus č. 3

Chybou bolo, že keď bol jeden blok určený zle, v nasledujúcom bloku som skúšal namapovať slová začínajúc takým, ktoré nezodpovedalo skutočnému slovu, ktoré naozaj malo prislúchať začiatku daného bloku. Cieľom je detekovať, že daný blok nie je namapovaný korektne a v takom prípade na nasledujúci blok neskúšať namapovať izolované slová len od slova, ktoré by malo nasledovať, ale skúsiť začať aj od nejakého skoršieho alebo neskoršieho slova

v poradí. Následne podľa toho upraviť aj predošlý blok (kde bola detekovaná veľká chyba DTW). Takouto úpravou vznikla finálna, v mojom nástroji používaná verzia algoritmu:

1. Vezmi nasledujúci blok b_k signálu súvisle nahovoreného textu.
2. Ak sa jedná o prvý blok b_1, alebo chyba namapovania pri predošlom bloku b_{k-1} vyšla menšia ako stanovená hranica E, pokračuj nasledujúcim krokom. V opačnom prípade opakuj kroky $3.$ $4.$ a $5.$ viackrát za sebou s tým, že skúšaj slová mapovať nie len od nasledujúceho slova s_n, ale vyskúšaj aj namapovania, kde je začiatočné slovo v bloku o niečo posunuté (s_{n+z} kde $z = -Z, \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots, Z$). Ku každému namapovaniu skús namapovať znovu aj predošlý blok s využitím slov, ktoré mu boli takto pridelené naviac alebo odobraté. Ako výsledné vezmi také namapovanie, pri ktorom dosiahneš priemernú chybu oboch blokov b_k a b_{k-1} najmenšiu.
3. Na základe pomerov dĺžok (analogicky ako v algoritme č. 1) odhadni počet q nasledujúcich izolovaných slov prislúchajúcich danému bloku b_k.
4. Usporiadaj si tieto izolované slová za sebou a použitím preskakovacieho DTW ich skús namapovať do pripraveného bloku b_k.
5. Toto namapovanie nevyskúšaj len pre odhadnutý počet slov, ale aj pre iné počty slov ($q + r$ kde $r = -R, \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots, R$) a ako výsledok zober to namapovanie, pri ktorom bola dosiahnutá najmenšia chyba.
6. Ak bola chyba predošlého bloku b_{k-1} menšia ako E pokračuj krokom $7.$, inak opakuj kroky $3.$ $4.$ a $5.$ s posunutým začiatočným slovom s_n, kým nevyskúšaš všetky možnosti určené parametrom Z (ako bolo spomínané v kroku $2.$). Potom pokračuj krokom $7.$
7. Vráť sa na krok $1.$ a rovnakým spôsobom pokračuj s nasledujúcim blokom b_{k+1}, až kým nemáš namapované všetky bloky.
8. Na konci pre každé slovo v rámci jemu prideleného úseku v príslušnom bloku, nájdi jeho presný začiatok a koniec pomocou slajdovacieho DTW.

Takáto úprava algoritmu výrazne obmedzila šírenie chyby na dlhých úsekoch. Problémom je, že skúšanie namapovania rôzneho počtu slov $q+r$ pre rôzne začiatočné slová s_{n+z} prináša výrazné zhoršenie časovej zložitosti algoritmu.

Kľúčovým problémom je teda vhodne zvoliť hraničnú chybu E, aby sa rôzne začiatočné slová neskúšali pre všetky bloky, ale len pre tie, kde je to naozaj potrebné. Hodnotu E môžeme meniť pomocou parametra

HRANICNA_CHYBA_BLOKU

ktorej defaultná hodnota je *10 000*. To znamená, že v praxi nikdy nie je prekročená, a skúšanie rôznych začiatočných slov a spätná oprava predošlého bloku sa defaultne nerobí. Je to z toho dôvodu, že presnú hodnotu nie je možné vo všeobecnosti vyčísliť nakoľko sa priemerná veľkosť chyby blokov rádovo mení podľa nastavenia jednotlivých parametrov. Preto odporúčam používať nástroj tak, že najprv sa nastaví hodnota hraničnej chyby veľká (rádovo *10 000*) a nechá sa zbehnúť algoritmus. Následne sa otvorí súbor *chybyBlokov.txt*, ktorý algoritmus vytvára ako sekundárny výstup v pracovnom adresári. V ňom sú po poradí zaznamenané chyby pri namapovaní jednotlivých blokov. Zhodnotí sa, ktoré bloky výraznejšie vyčnievajú nad priemer a hodnota hraničnej chyby sa zvolí podľa nich (niekde medzi priemerom a tými, čo vyčnievajú).

Okrem hraničnej chyby ešte máme možnosť nastavovať hodnotu R pomocou parametra

POLOMER_PREHLADAVANIA_SLOV

ktorá nám hovorí, plus-mínus koľko slov sa snažím na daný blok namapovať okrem odhadu. Táto hodnota nie je vyčíslená absolútne, ale zadáva sa ako násobok odhadovaného počtu slov q daného bloku. Defaultná hodnota je *0.5* čo znamená, že sa na každý blok skúša namapovať odhadnutý počet slov plus-mínus 50% z neho. Pozorovaním výpočtu na mnou testovaných vstupoch som zistil, že 50% je optimálne nastavenie aby v danom rozsahu s veľkou pravdepodobnosťou ležal aj skutočný počet slov, ktorý prislúcha danému bloku.

Problémom je, že ak sa na daný blok odhadne podľa dĺžok veľmi malý počet slov (napríklad jedno), pri 50% tolerancii sa naň skúša namapovať buď jedno alebo dve slová (minimálny počet je vždy jedno a zaokrúhľuje sa nahor). Mohlo by sa ale stať, že tam mali byť tri slová. Tento problém som vyriešil pridaním parametra

MINIMALNY_POLOMER_PREHLADAVANIA_SLOV

ktorého pokusne určená defaultná hodnota je *3* a znamená, plus-mínus koľko slov sa bude minimálne skúšať namapovať na každý blok bez ohľadu na odhad.

Podobne aj pre hodnotu Z som zaviedol obdobné parametre. Pomocou parametra

POLOMER_HLADANIA_ZACIATKU_BLOKU

nastavujem plus-mínus koľko slov sa bude posúvať začiatočné slovo mapovaného bloku a následne upravovať počet slov v predošlom bloku, pri ktorom vyšla chyba väčšia ako E. Defaultná hodnota je rovnako ako v predošlom prípade *0.5* (50% slov predošlého bloku). Rovnako parameter

MINIMALNY_POLOMER_HLADANIA_ZACIATKU_BLOKU

s defaultnou hodnotou *5* slúži na zabezpečenie, aby sa prešlo aspoň určitý minimálny počet slov, ak by počet slov predošlého bloku vyšiel príliš malý (jedno, dve, ...). Parameter, ktorý stojí tiež za povšimnutie je

DOPASOVANIE_SLOV

Hovorí nám do akej vzdialenosti (vyjadrenej ako násobok dĺžky príslušného slova) je vo finálnej fáze skúšaný algoritmus slajdovacie DTW na nájdenie presného začiatku a konca slova. Je dôležité aby táto hodnota nebola príliš veľká lebo môže dochádzať k výrazným chybám. Defaultná hodnota je *0.1* (presný začiatok a koniec je hľadaný do vzdialenosti 10% pôvodnej dĺžky slova). Ak nechceme po namapovaní blokov použiť slajdovacie DTW nastavíme ho na hodnotu *0*.

4.6 Vylepšenia

Tak ako pri mnohých iných algoritmoch, ani pri tomto sa nezaobídeme bez vhodných predpokladov, očakávaní a obmedzení, ktoré som skúšal do algoritmu zakomponovať s cieľom dosahovania lepších výsledkov. V niektorých prípadoch som bol vcelku úspešný v iných menej.

4.6.1 Odhad počtu slov

Zistil som, že nie je vždy dobré priraďovať rovnakú váhu chybe namapovania rôzneho počtu slov na blok. Rozhodol som sa zvýhodňovať namapovania tých počtov slov, ktoré sú bližšie k odhadom počítaným z pomerov dĺžok. Toto môže byť realizované dvomi spôsobmi:

1. Zvýhodňované sú počty bližšie k odhadu z celkového počtu slov a celkovej dĺžky všetkých blokov. Odhad počtu slov q pre blok b_k kde $k = 1, \ldots, K$ sa počíta zo vzťahu:

$$\frac{l(b_k)}{\sum_{i=1}^{K} l(b_i)} = \frac{l(s_{1_k}) + l(s_{2_k}) + \ldots + l(s_{q_k})}{\sum_{i=1}^{N} l(s_i)}$$

   kde K je celkový počet blokov a N je celkový počet slov.

2. Zvýhodňované sú počty bližšie k odhadu z ostávajúceho počtu slov a ostávajúcej dĺžky nenamapovaných blokov. Odhad počtu slov q pre blok b_k kde $k = 1, \ldots, K$ sa počíta z podobného vzťahu, ako bol predchádzajúci:

$$\frac{l(b_k)}{\sum_{i=k}^{K} l(b_i)} = \frac{l(s_{1_k}) + l(s_{2_k}) + \ldots + l(s_{q_k})}{\sum_{i=s_{1_k}}^{N} l(s_i)}$$

   rozdiel je len v rozsahoch súm v menovateľoch. Tento odhad oproti predošlému predpokladá, že to čo už je namapované je v poriadku a pre nasledujúce bloky sa bude robiť odhad už len z nasledujúcich (ešte nenamapovaných) blokov a slov.

Úprava výslednej chyby σ sa pre obidva prípady odhadov vykonáva podľa vzťahu:

$$\sigma = \sigma' + v(2^{|q-h|} - 1)\sigma'$$

kde σ' je pôvodná chyba pred úpravou, q je odhadovaný počet slov pri príslušnom type odhadu, h je práve skúšaný počet slov a v je parameter ovplyvňujúci veľkosť zmeny chyby. Daný vzťah som vyvodil z pozorovaní a pokusov, ktoré ukázali, že lineárny nárast chyby pre zvyšujúci sa rozdiel mapovaného počtu slov od odhadu nie je postačujúci. Používajú sa vždy obidva zvýhodnenia súčasne (t.j. chyba každého namapovania určitého počtu slov sa upraví najprv podľa jedného a následne podľa druhého odhadu). Hodnotu v je možné nastaviť pomocou parametra

VAHA_CHYBY

ktorého defaultnú hodnotu som pokusne určil ako *0.005*. Po dosadení do uvedeného vzťahu dostávame nárast chyby pre počet slov rovný odhadu 0%, pre rozdiel o jedno slovo 0.5%, dve slová 1.5%, tri slová 3.5%, ...

4.6.2 Odhad dĺžky slov

Pri preskakovacom DTW som sa stretával s problémom, keď pri mapovaní určitého počtu slov na blok bol niektorým slovám priradený veľmi krátky úsek daného bloku. To ma podnietilo k tomu, aby som aj v takomto prípade začal počítať (z pomerov dĺžok), aké dlhé úseky bloku by mali jednotlivým slovám po výpočte zodpovedať. Ak je výsledok rozdielny, znevýhodňujem takéto namapovanie zvýšením chyby σ_s (ktorá sa následne premietne do celkovej chyby bloku) príslušného slova s podľa vzťahu:

$$\sigma_s = \sigma'_s + \frac{|l(b_s) - l^*(b_s)|}{l^*(b_s)}\sigma'_s$$

kde σ'_s je chyba pred úpravou, $l(b_s)$ je skutočná dĺžka úseku bloku priradeného danému slovu a $l^*(b_s)$ je odhadovaná dĺžka úseku bloku priradeného danému slovu. Daný vzťah hovorí, že chyba sa zhorší o toľko percent, o koľko sa líši skutočná dĺžka od odhadu. Pokusy ukázali, že je to veľmi dobrá stratégia. Pomocou nej som obmedzil vznik namapovaní, pri ktorých je slovu priradený nepatrne krátky (nulový) úsek bloku.

4.6.3 Globálne ohraničenie

Pri algoritme DTW sa ukázalo, že globálne ohraničenia ohýbacej cesty môžu mať priaznivý vplyv na nachádzanie dobrých namapovaní dvoch postupností. Implementoval globálne ohraničenia presne podľa postupu spomínanom v časti 3.1 (globálne obmedzenia). Používateľ má možnosť nastavovať polomer prípustnej oblasti pomocou parametra

GLOBALNE_OHRANICENIE_DTW

ktorého hodnota sa zadáva v sekundách. Bohužiaľ som nepozoroval zlepšovanie (skôr naopak) výsledkov pri testovaní rôznych šírok prípustnej oblasti. Preto som defaultnú hodnotu tohto parametra nastavil na *20.0* (aby to v praxi nemalo na výpočet žiaden vplyv).

4.6.4 Zmena tempa

Pri počúvaní nahrávok jednotlivých príbehov som si všimol, že slová nahovorené izolovane majú dobu trvania vo väčšine prípadov výrazne dlhšiu, ako

súvisle nahovorené tie isté slová. Rozmýšľal som, či by nebolo vhodné upraviť tempo jednej z nahrávok, aby si dĺžky rovnakých slov oboch nahrávok lepšie zodpovedali. Nakoniec to neprinieslo zlepšenie, takže som to vo svojom algoritme nepoužil. Chyba bola asi v tom, že aj keď boli trvania korešpondujúcich slov oboch príbehov rozdielne, táto rozdielnosť nebola rovnomerná, ale vznikala hlavne v dôsledku zdôrazňovania (pri nahrávke izolovaných slov) dlhých samohlások, zatiaľ čo spoluhlásky boli vyslovované zhruba rovnako dlho. Preto rovnomerná zmena tempa nepriniesla žiadané zlepšenie.

4.7 Metóda merania chyby

Za dôležité považujem ešte spomenúť, ako som vyhodnocoval úspešnosť pri pokusoch a testoch môjho algoritmu. Musel som si ručne spracovať (prejsť a zaznačovať čo najpresnejšie časy začiatkov a koncov slov) testovacie príbehy. Výsledky som zaznamenal do testovacích súborov. Po skončení samotného algoritmu sa spustí testovací proces, ktorý porovnáva vypočítané časy s hodnotami v testovacom súbore. Výstupom tohto procesu sú 4 údaje:

1. *Priemerná chyba na slovo* - $\overline{\Sigma}$ - táto hodnota sa získa ako súčet rozdielov vypočítaných začiatkov a koncov od hodnôt ich začiatkov a koncov uvedených v testovacom súbore a následným vydelením počtom slov. Jej hodnota je v sekundách.

2. *Maximálna chyba na slovo* - Σ_{MAX} - určuje súčet rozdielu začiatku a konca najhoršie namapovaného slova od príslušných hodnôt v testovacom súbore (v sekundách).

3. *Priemerná chyba krát intenzita na slovo* - $\overline{\phi}$ - Časový rozdiel začiatkov a koncov slov od hodnôt v testovacom súbore nemusí byť vždy tým najvhodnejším kritériom. V prípade, že je slovu priradený určitý úsek signálu a v tomto úseku sa nachádza samotné slovo plus ešte navyše určité ticho na začiatku a na konci, ktoré sa v ručnom vyznačovaní odseklo, je toto slovo určené správne, aj keď jeho začiatok a koniec nezodpovedajú začiatku a koncu v testovacom súbore. Preto som sa rozhodol zaviesť tento alternatívny typ chyby, ktorý vypočíta časový rozdiel začiatkov a koncov od začiatkov a koncov uvedených v testovacom súbore a tieto hodnoty vynásobí priemernou intenzitou signálu na daných úsekoch rozdielu.

Ak je hodnota pre slovo nízka, značí to, že v daných časových úsekoch rozdielu je ticho, a teda je slovo určené správne.

Hodnota predstavuje priemer počítaný zo všetkých slov.

4. *Chyba krát intenzita najhoršie určeného slova* - ϕ_{MAX} - Ako už názov napovedá, hodnota zachytáva chybu časovo najhoršie určeného slova (to, ktorého časová chyba je Σ_{MAX}) pomocou takéhoto alternatívneho vyčísľovania chyby.

Je treba si uvedomiť, že chyba je počítaná ako rozdiel začiatkov plus rozdiel koncov. Teda chyba zaznamenávajúca posun slova oproti miestu kde by skutočne malo byť sa približne vypočíta ako polovica tejto hodnoty.

Rovnako treba brať ohľad na to, že pri ručnom spracovávaní súvisle nahovoreného príbehu, sa tiež nedajú vždy presne určovať časy začiatkov a koncov slov. Pár stotín sekundy sa tam občas kľudne môže stratiť.

Názov testovacieho súboru (ktorý sa má nachádzať v pracovnom adresári) ako sekundárneho vstupu algoritmu sa dá zmeniť pomocou parametra

TEST

ktorého defaultná hodnota je *suvisle-handwork.txt*. Formát testovacieho súboru je rovnaký ako formát výstupu. Ak sa testovací súbor nenachádza v pracovnom adresári, alebo má zlý formát, algoritmus zobrazí varovné hlásenie.

4.8 Ostatné parametre

Medzi doteraz nespomenuté, ale užitočné parametre patrí aj parameter

DLZKA_OKIENKA

ktorý hovorí, na aké veľké časti (mikrosegmety) sa "naokienkuje" signál, aby sa následne mohli z týchto častí vypočítať príslušné vektory príznakov. Hodnota tohoto parametra by mala byť určovaná podľa hodnoty vzorkovacej frekvencie vstupov (sample rate) tak, aby (ako už bolo spomínané) dĺžka jedného mikrosegmentu bola približne 10-30 ms. Hodnota býva spravidla tvaru 2^k kde k je kladné celé číslo, kvôli efektívnemu počítaniu rýchlej Fourierovej transformácie potrebnej na výpočet MFCC koeficientov. Defaultná hodnota tohto parametra je *512*. To pri vzorkovacej frekvencii 16 000 Hz zodpovedá dĺžke jedného

mikrosegmentu 32 ms. Pre vzorkovaciu frekvenciu 22 050 Hz je to približne 23 ms. K tomuto parametru sa viaže parameter

PREKRYV_OKIENOK

určujúci akou veľkou časťou sa budú jednotlivé mikrosegmenty prekrývať. Defaultná hodnota je *256*. Obidva tieto parametre výraznou mierou ovplyvňujú čas behu algoritmu, preto treba dobre premyslieť, aké hodnoty im priraďujeme.

Kapitola 5

Experiment

V tejto kapitole sa venujem testovaniu vytvoreného algoritmu a vyhodnoteniu dosiahnutých výsledkov. Ako testovacie vstupy som použil dva príbehy, pre ktoré som posluchom vytvoril testovacie súbory obsahujúce začiatky a konce jednotlivých slov v súvisle nahovorenej nahrávke.

Pri konkrétnych hodnotách netreba zabúdať na to, že takto vytvorené testovacie súbory môžu obsahovať malé nepresnosti, ktoré som posluchom nezachytil.

5.1 Testovanie

1. Pre príbeh *"Vybrané slová v Bytči"* boli sledované 4 typy chyby nasledovné:

   $\overline{\Sigma} = 0,126s$ $\quad \Sigma_{MAX} = 1,875s$
   $\overline{\phi} = 160,410$ $\quad \phi_{MAX} = 1371,395$

   Na výpočet boli použité defaultné parametre. Nastavenie hraničnej chyby bloku výsledok v tomto prípade nezlepšilo.

   Keď som zmenil spôsob počítania vzdialeností na

   POCITANIE_VZDIALENOSTI = -1

   dosiahnuté výsledky sa v priemere o niečo zhoršili:

   $\overline{\Sigma} = 0,154s$ $\quad \Sigma_{MAX} = 1,463s$
   $\overline{\phi} = 175,555$ $\quad \phi_{MAX} = 1145,914$

   Menšia chyba v prvom prípade je pravdepodobne spôsobená tým, že intenzita šumu v pozadí je v celom príbehu veľmi nízka. Teda váhovanie MFCC pri počítaní vzdialenosti pravdepodobne spôsobí presnejšie namapovanie oproti druhému prípadu, kde sa prejavili hlavne zložky signálu z nižšou frekvenciou.

2. Pre príbeh s názvom *"Baltíkove čary"* som dosiahol pre defaultné hodnoty parametrov nasledovné výsledky:

   $\overline{\Sigma} = 1,352s$ $\quad \Sigma_{MAX} = 7,799s$
   $\overline{\phi} = 3282,172$ $\quad \phi_{MAX} = 18101,656$

   S týmito výsledkami som nebol spokojný. Po nastavení hraničnej chyby bloku na

   HRANICNA_CHYBA_BLOKU = 20

   boli dosiahnuté výsledky výrazne lepšie:

   $\overline{\Sigma} = 0,325s$ $\quad \Sigma_{MAX} = 3,175s$
   $\overline{\phi} = 660,406$ $\quad \phi_{MAX} = 7049,293$

Keď som použil počítanie vzdialeností

POCITANIE_VZDIALENOSTI = -1

(s defaultnou hraničnou chybou bloku), výsledky sa ešte viac zlepšili:

$\overline{\Sigma} = 0,139s$ $\quad \Sigma_{MAX} = 1,371s$
$\overline{\phi} = 179,397$ $\quad \phi_{MAX} = 401,197$

a nastavenie hraničnej chyby bloku na hodnotu

HRANICNA_CHYBA_BLOKU = 500

Výsledky ešte o niečo zlepšilo:

$\overline{\Sigma} = 0,129s$ $\quad \Sigma_{MAX} = 1,371s$
$\overline{\phi} = 154,435$ $\quad \phi_{MAX} = 401,197$

Rozdiel, prečo boli v tomto prípade pri nepoužití váhovania MFCC výsledky výrazne lepšie oproti prvému príbehu je pravdepodobne v tom, že intenzita šumu v pozadí druhého príbehu je väčšia. Preto pri váhovaní dochádzalo k výraznejšiemu presadzovaniu zložiek signálu s vyššou frekvenciou (šumu), čo malo za následok horšie namapovanie.

Z oboch príkladov (pri najlepších nastaveniach parametrov) vidieť, že priemerné posunutie slova oproti pozícii, kde by sa malo nachádzať je približne 6 stotín sekundy, čo nie je veľa. Keď do toho započítame, najhoršie určené slovo (ktorého chyba je pomerne veľká), ľahko vyvodíme, že väčšina slov bola určená veľmi presne.

Reálny čas behu výpočtov (obidve nahrávky obsahovali niečo nad 100 slov) bol o niečo kratší ako 1 minúta (na procesore AMD Athlon 64 2800+). Určené hranice všetkých slov pre obidva príbehy porovnané s ich skutočnou (posluchom určenou) pozíciou sú k dispozícii v dodatku A.

5.2 Chybne určené slová

V oboch testovacích príkladoch sa vyskytli slová, ktoré boli určené výrazne horšie, oproti ostatným. Poďme sa skúsiť zamyslieť, prečo to tak bolo.

V prvom príbehu (*"Vybrané slová v Bytči"*) to boli slová:

č.	**slovo**	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	$\mathbf{\Sigma}$	ϕ
47	sa	20,780	21,080	20,794	21,073	0,021	13,478
48	kde	21,410	21,540	21,371	21,939	0,438	831,717
49	je	21,540	21,710	22,493	22,633	1,876	1371,396
50	trh	21,810	21,990	22,633	22,633	1,466	719,325
51	bytčan	22,520	22,940	22,656	22,946	0,142	171,138

V stĺpci ″**č.**″ sa nachádza poradové číslo slova, v stĺpci ″**slovo**″ je príslušné slovo, stĺpce ″$\mathbf{Z_1}$″ a ″$\mathbf{K_1}$″ uvádzajú skutočný (posluchom určený) začiatok a koniec slova a v stĺpcoch ″$\mathbf{Z_2}$″ a ″$\mathbf{K_2}$″ sa nachádza začiatok a koniec slova určený pomocou môjho algoritmu. Údaje sú uvedené v sekundách. V stĺpci ″$\mathbf{\Sigma}$″ je zaznamenaná chyba slova (rozdiel začiatkov plus rozdiel koncov) v sekundách a v stĺpci ″ϕ″ je chyba počítaná ako časový rozdiel začiatkov a koncov vynásobený priemernou intenzitou na intervaloch tohto rozdielu.

Chyba vznikla pravdepodobne preto, že slová "kde je"v súvisle nahovorenom texte boli vyslovené rýchlo za sebou (hláska ″j″ je len nepatrná) a algoritmus na ne namapoval samotné izolované slovo "kde". Následne sa chyba krátky čas šírila.

Chybne určený bol tiež úsek:

č.	**slovo**	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	$\mathbf{\Sigma}$	ϕ
53	povedal	23,290	23,890	23,295	23,852	0,043	18,409
54	choďte	24,200	24,650	24,271	24,944	0,365	493,188
55	za	24,650	24,830	25,014	25,142	0,676	1045,776
56	starobylý	24,830	25,830	25,200	25,804	0,396	714,472
57	dom	26,180	26,480	26,112	26,495	0,083	60,446

V tomto prípade bolo slovo ″choďte″ mierne natiahnuté. Pravdepodobne to bolo spôsobené kombináciou spoluhlások ″ďť″, ktoré sa namapovali aj na spoluhlásku ″z″ slova ″za″. Slovo ″za″ bolo vyslovené nevýrazne (v súvislom nahovorení) a algoritmus mu priradil úsek signálu zodpovedajúci vysloveniu ″sta″ (začiatok slova ″starobylý″), na ktorý sa zvukom podobá.

V druhom príbehu (*″Baltíkove čary″*) vznikla prvá väčšia chyba hneď na začiatku:

č.	slovo	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	$\mathbf{\Sigma}$	ϕ
1	baltíkove	0,820	1,580	0,902	1,862	0,364	806,288
2	čary	1,600	1,930	2,201	2,249	0,920	880,863
3	raz	2,230	2,420	2,249	2,425	0,024	14,972

Myslím si, že to bolo spôsobené výraznými odlišnosťami vo vyslovení prvých dvoch slov (”baltíkove čary”) v súvisle nahovorenom príbehu a v príbehu nahovorenom po slovách. V súvisle nahovorenom príbehu bolo slovo ”baltíkove” krátke s málo výraznými samohláskami, zatiaľ čo v izolovane nahovorenom príbehu bolo vyslovené s výraznými samohláskami. Pri slove ”čary” to bolo opačne. V súvisle nahovorenom príbehu bolo toto slovo vyslovené pomerne zreteľne, zatiaľ čo pri izolovane nahovorenej nahrávke bolo málo výrazné.

Druhý chybný úsek:

č.	slovo	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	$\mathbf{\Sigma}$	ϕ
11	kvety	5,100	5,420	5,097	5,417	0,006	6,902
12	do	5,420	5,600	5,449	5,801	0,230	197,364
13	hory	5,600	5,840	6,382	6,430	1,372	401,197
14	peknú	6,410	6,780	6,446	6,718	0,098	184,364

bol spôsobený veľmi nevýrazným vyslovením slova ”hory” v súvisle nahovorenom príbehu. Izolované slovo ”do” bolo roztiahnuté na úkor slova ”hory” a slovo hory sa presunulo až do nasledovného bloku kde sa namapovalo na krátky úsek signálu na začiatku.

V treťom prípade:

č.	slovo	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	$\mathbf{\Sigma}$	ϕ
31	zrazu	13,970	14,310	14,000	14,384	0,104	124,405
32	stalo	14,310	14,920	14,480	15,248	0,498	275,997
33	baltík	15,520	16,030	15,526	16,006	0,030	33,692

bolo v slove ”stalo” vyslovené nevýrazné ”s” (veľmi podobné šumu v pozadí) na začiatku preto bol začiatok posunutý. Koniec slova je síce posunutý, ale po skontrolovaní nahrávky (a tiež podľa nízkej chyby ϕ) som zistil, že na predĺženom úseku signálu za slovom je zaznamenané len ticho.

Štvrtá chyba:

č.	**slovo**	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	$\mathbf{\Sigma}$	ϕ
54	vyzeral	27,480	27,990	27,493	27,941	0,062	75,769
55	hrozivo	27,990	28,650	28,005	28,405	0,260	505,566
56	z	28,910	29,030	28,453	28,661	0,826	583,181
57	očí	29,030	29,410	28,953	29,353	0,134	143,610

bola spôsobená namapovaním spojky ″z″ na zvuk ″vo″ (koniec slova ″hrozivo″). To bolo pravdepodobne spôsobené veľmi výrazným vyslovením spojky ″z″ v izolovanej nahrávke (akoby za ňou bol náznak samohlásky ″o″).

Posledná chyba:

č.	**slovo**	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	$\mathbf{\Sigma}$	ϕ
88	vznášal	44,890	45,270	44,887	45,287	0,020	24,281
89	sa	45,270	45,510	45,335	45,655	0,210	488,513
90	nad	45,510	45,700	45,703	46,135	0,628	1472,797
91	hradným	46,160	46,840	46,231	46,887	0,118	205,748

bola spôsobená chybným nahovorením nahrávky. Pri čítaní slova ″hradným″ sa čitateľ mierne zakoktal a vyslovil slovo ″hrad hradným″. Slovo ″nad″ sa preto namapovalo na prvú časť tohto zvuku na ktorú sa zvukom podobá.

Kapitola 6

Záver

Očakávaným záverom je vyhodnotenie cieľa práce. Myslím si, že moja práca tento svoj cieľ splnila. Podrobne som preštudoval algoritmus DTW, navrhol som a otestoval rôzne jeho modifikácie a ich kombináciou som vytvoril nástroj, ktorý rieši zadaný problém.

Výpočet hraníc slov nie je 100% bez chybný, ale myslím si, že je prijateľný. Nástroj sa určite bude môcť použiť na vytvorenie funkcie "karaoke" v Multimediálnej čítanke s tým, že niekoľko chybne určených slov bude musieť používateľ opraviť.

Výpočtová náročnosť algoritmu pri správnom zaobchádzaní je tiež prijateľná.

Algoritmus funguje aj pre nekonzistentné vstupy, čo potvrdzuje aj posledný chybný úsek príbehu *"Baltíkove čary"* popísaný v časti 5.2. Algoritmus síce nedetekuje chybnú časť, ale vzniknutá chyba sa šíri len krátkodobo (v závislosti na dĺžke nekonzistentného úseku a nastavení parametrov) a nepokazí celý výpočet. Riešenie problému presnej detekcie chybných úsekov nechávam na svojich možných nasledovníkov.

Dúfam, že moja práca prispeje do oblasti výskumu rozpoznávania reči a pomôže aspoň trochu urýchliť jej vývoj.

Literatúra

[1] "Laboratory of acoustics and audio signal processing." http://www.acoustics.hut.fi/.

[2] P. Jozef, *Komunikace s počítačem mluvenou řečí*. Praha: Academia, 1995.

[3] P. Sliacky, "Využitie skrytých markovových modelov pre počítačové rozpoznávanie hovorenej českej reči," Master's thesis, FJFI CVUT Praha, 2006.

[4] M. Nagy, "Html verzia multimediálnej čítanky." http://cpr.ii.fmph.uniba.sk/citanka/.

[5] C. Lévy, G. Linares, and P. Nocera, "Comparison of several acoustic modeling techniques and decoding algorithms for embedded speech recognition systems." http://www.lia.univ-avignon.fr/fich_art/419-article.pdf.

[6] S. B. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," pp. 65–74, 1990.

[7] M. Nagy, "Skryté markovove modely a rozpoznávanie číslic," Master's thesis, FMFI UK Bratislava, 2004. http://www.ii.fmph.uniba.sk/~mnagy/documents/Nagy2004_rigo.pdf.

[8] O. Seman, "Ovládanie počítača hlasom," Master's thesis, FMFI UK Bratislava, 2004. http://delo.dcs.fmph.uniba.sk/~seman/dipl/diplomka.pdf.

[9] F. J. Owens, R. Andonie, G. H. Zheng, A. Cataron, and S. Manciulea, "A comparative study of the multi-layer perceptron, the multi-output

layer perceptron, the time-delay neural network and the kohonen self-organising map in an automatic speech recognition task," in *Proceedings of the EIS'98 International ICSC Symposium on Engineering of Intelligent Systems*, 1998. http://www.cwu.edu/~andonie/MyPapers/Tenerife_speech.pdf.

[10] E. Choubassi, M. E. Khoury, H. Alagha, C. Skaf, and J. Al-Alaoui, "Arabic speech recognition using recurrent neural networks," in *Signal Processing and Information Technology*, pp. 543–547, 2003.

[11] E. Keogh and M. Pazzani, "Derivative dynamic time warping," 2001.

[12] "The htk book." http://htk.eng.cam.ac.uk/prot-docs/htkbook.pdf.

[13] D. McEnnis, C. McKay, I. Fujinaga, and P. Depalle, "Jaudio: A feature extraction library," *Proceedings of the 2005 International Conference on Music*, 2005. http://ismir2005.ismir.net/proceedings/2103.pdf.

[14] M. Nagy, "Experimentálny systém rozpoznávania reči," Master's thesis, FMFI UK Bratislava, 1999. http://www.ii.fmph.uniba.sk/~mnagy/documents/Nagy1999_dipl.pdf.

[15] D. McEnnis, C. McKay, I. Fujinaga, and P. Depalle, *jAudioFeatureExtractor*. http://www.music.mcgill.ca/~mcennis/doc/.

[16] "Wikipedia - the free encyclopedia." http://www.wikipedia.org/.

[17] "Vznik a složení lidského hlasu." http://www.info.estranky.cz/clanky/biologie/vznik-a-slozeni-lidskeho-hlasu.

Dodatok A

Priložené výsledky

V nasledujúcej prílohe čitateľovi poskytujem výsledky, ktoré som dosiahol pri experimente na dvoch testovacích príbehoch spomínaných v kapitole 5. Výsledky sú zoradené v prehľadnej tabuľke. V stĺpci "**č.**" sa nachádza poradové číslo slova, v stĺpci "**slovo**" je príslušné slovo, stĺpce "$\mathbf{Z_1}$" a "$\mathbf{K_1}$" uvádzajú skutočný (posluchom určený) začiatok a koniec slova a v stĺpcoch "$\mathbf{Z_2}$" a "$\mathbf{K_2}$" sa nachádza začiatok a koniec slova určený pomocou môjho algoritmu. Údaje sú uvedené v sekundách. V stĺpci "$\mathbf{\Sigma}$" je zaznamenaná chyba slova (rozdiel začiatkov plus rozdiel koncov) v sekundách a v stĺpci "ϕ" je chyba počítaná ako časový rozdiel začiatkov a koncov vynásobený priemernou intenzitou na intervaloch tohto rozdielu.

A.1 "Vybrané slová v Bytči"

č.	**slovo**	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	$\mathbf{\Sigma}$	ϕ
1	vybrané	0,260	0,730	0,267	0,697	0,040	85,646
2	slová	0,730	1,150	0,767	1,103	0,084	176,748
3	v	1,150	1,280	1,127	1,219	0,084	101,008
4	bytči	1,280	1,740	1,254	1,719	0,047	38,374
5	vybrali	2,150	2,610	2,163	2,581	0,042	53,775
6	sa	2,610	2,800	2,627	2,871	0,088	117,293
7	raz	2,800	3,070	2,894	3,057	0,107	149,214
8	vybrané	3,110	3,660	3,115	3,637	0,028	47,906
9	slová	3,660	4,030	3,719	4,079	0,108	136,101

č.	**slovo**	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	$\mathbf{\Sigma}$	ϕ
10	do	4,120	4,120	4,125	4,299	0,184	230,094
11	bytče	4,310	4,620	4,346	4,775	0,191	347,707
12	na	4,620	4,830	4,810	4,880	0,240	385,776
13	trh	4,910	5,090	4,891	5,089	0,020	0,450
14	išli	5,600	5,890	5,602	5,869	0,023	10,805
15	cez	5,920	6,120	5,904	6,159	0,055	7,697
16	bystrú	6,190	6,570	6,217	6,624	0,081	69,322
17	bystrinu	6,670	7,270	6,705	7,239	0,066	80,952
18	išli	7,700	8,010	7,705	8,007	0,008	2,275
19	cez	8,060	8,250	8,042	8,274	0,042	9,472
20	les	8,250	8,530	8,297	8,518	0,059	26,608
21	kde	8,600	8,690	8,541	8,715	0,084	28,866
22	sa	8,690	8,940	8,738	8,912	0,076	83,823
23	hmýril	8,940	9,330	8,947	9,307	0,030	53,734
24	hmyz	9,330	9,740	9,377	9,737	0,050	101,461
25	a	9,960	10,000	9,940	10,010	0,030	3,072
26	cez	10,010	10,250	10,021	10,184	0,077	79,281
27	lúku	10,250	10,690	10,242	10,741	0,059	72,080
28	kde	10,690	10,920	10,788	10,915	0,103	112,444
29	rástli	10,920	11,400	10,950	11,356	0,074	98,857
30	samé	11,400	11,860	11,438	11,902	0,080	83,788
31	byliny	11,950	12,420	11,983	12,413	0,040	50,588
32	zrazu	12,980	13,410	13,025	13,396	0,059	29,470
33	sa	13,410	13,610	13,443	13,663	0,086	209,527
34	mykli	13,610	14,130	13,710	14,116	0,114	342,947
35	pred	14,460	14,660	14,468	14,712	0,060	1,940
36	pyskom	14,720	15,120	14,758	15,118	0,040	37,435
37	im	15,180	15,340	15,165	15,374	0,049	6,655
38	preletel	15,390	16,020	15,432	15,978	0,084	71,034
39	výr	16,020	16,330	16,059	16,326	0,043	33,124
40	keď	16,880	17,090	16,878	17,052	0,040	42,932
41	vošli	17,090	17,560	17,098	17,551	0,017	20,006

č.	**slovo**	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	$\mathbf{\Sigma}$	ϕ
42	medzi	17,560	17,980	17,621	17,980	0,061	121,178
43	obydlia	18,030	18,540	18,050	18,468	0,092	152,307
44	obyvateľov	18,560	19,280	18,573	19,281	0,014	5,568
45	bytče	19,350	19,790	19,385	19,792	0,037	35,797
46	opýtali	20,130	20,780	20,155	20,713	0,092	110,705
47	sa	20,780	21,080	20,794	21,073	0,021	13,478
48	kde	21,410	21,540	21,371	21,939	0,438	831,717
49	je	21,540	21,710	22,493	22,633	1,876	1371,396
50	trh	21,810	21,990	22,633	22,633	1,466	719,325
51	bytčan	22,520	22,940	22,656	22,946	0,142	171,138
52	im	23,010	23,230	22,981	23,225	0,034	10,204
53	povedal	23,290	23,890	23,295	23,852	0,043	18,409
54	choďte	24,200	24,650	24,271	24,944	0,365	493,188
55	za	24,650	24,830	25,014	25,142	0,676	1045,776
56	starobylý	24,830	25,830	25,200	25,804	0,396	714,472
57	dom	26,180	26,480	26,112	26,495	0,083	60,446
58	tam	26,540	26,790	26,553	26,820	0,043	128,632
59	je	26,790	26,930	26,843	26,994	0,117	215,409
60	trh	27,030	27,220	27,018	27,238	0,030	6,151
61	poslúchli	27,760	28,450	27,780	28,314	0,156	216,045
62	bytčana	28,450	29,070	28,430	28,999	0,091	98,977
63	a	29,070	29,240	29,069	29,219	0,022	26,166
64	išli	29,240	29,620	29,254	29,603	0,031	33,741
65	za	29,620	29,780	29,637	29,788	0,025	21,514
66	starobylý	29,780	30,510	29,858	30,450	0,138	184,861
67	dom	30,590	30,840	30,531	30,845	0,064	47,505
68	zrazu	31,340	31,780	31,366	31,714	0,092	109,378
69	uvideli	31,840	32,390	31,796	32,330	0,104	69,510
70	trh	32,450	32,720	32,411	32,690	0,069	10,761
71	všetci	33,080	33,480	33,107	33,490	0,037	9,250
72	od	33,650	33,810	33,537	33,734	0,189	143,938
73	radosti	33,810	34,400	33,804	34,373	0,033	32,083

č.	**slovo**	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	$\mathbf{\Sigma}$	ϕ
74	vykríkli	34,400	35,110	34,489	35,093	0,106	131,627
75	hurá	35,400	36,050	35,438	36,019	0,069	59,842
76	vošli	36,680	37,090	36,688	37,106	0,024	9,717
77	do	37,130	37,240	37,153	37,338	0,121	202,924
78	stánku	37,240	37,950	37,396	37,942	0,164	281,551
79	a	37,950	38,020	38,000	38,035	0,065	147,941
80	kúpili	38,100	38,630	38,081	38,592	0,057	86,360
81	si	38,630	38,840	38,650	38,906	0,086	273,746
82	mydlo	38,840	39,360	38,964	39,358	0,126	499,991
83	lebo	39,930	40,190	39,920	40,199	0,019	11,290
84	boli	40,230	40,560	40,246	40,536	0,040	60,754
85	hladní	40,560	41,030	40,594	41,012	0,052	79,052
86	zjedli	41,500	42,010	41,529	41,970	0,069	56,999
87	to	42,090	42,280	42,028	42,283	0,065	2,545
88	ale	42,350	42,550	42,330	42,597	0,067	113,473
89	nechutilo	42,550	43,200	42,666	43,166	0,150	318,476
90	im	43,210	43,490	43,235	43,491	0,026	6,423
91	to	43,550	43,700	43,526	43,711	0,035	5,261
92	tak	43,950	44,090	43,931	44,082	0,027	6,164
93	zistili	44,090	44,750	44,093	44,302	0,451	948,931
94	že	44,750	44,950	44,349	44,802	0,549	1304,072
95	lepšie	44,950	45,390	44,883	45,347	0,110	364,668
96	chutia	45,390	45,830	45,429	45,823	0,046	25,413
97	byliny	45,940	46,420	45,905	46,392	0,063	117,461
98	a	46,470	46,580	46,450	46,601	0,041	5,466
99	pýr	46,700	46,970	46,636	46,950	0,084	6,421
100	lebo	47,220	47,450	47,199	47,454	0,025	8,766
101	sú	47,450	47,650	47,489	47,710	0,099	162,682
102	bylinožravce	47,740	48,640	47,803	48,580	0,123	194,570

A.2 "Baltíkove čary"

č.	slovo	Z_1	K_1	Z_2	K_2	Σ	ϕ
1	baltíkove	0,820	1,580	0,902	1,862	0,364	806,288
2	čary	1,600	1,930	2,201	2,249	0,920	880,863
3	raz	2,230	2,420	2,249	2,425	0,024	14,972
4	sa	2,420	2,550	2,425	2,585	0,040	15,821
5	baltík	2,610	3,100	2,617	3,001	0,106	223,814
6	rozhodol	3,100	3,720	3,081	3,641	0,098	278,468
7	že	3,720	3,850	3,689	3,897	0,078	108,224
8	si	3,850	4,020	3,897	4,105	0,132	75,950
9	pôjde	4,120	4,490	4,121	4,457	0,034	71,809
10	natrhať	4,490	5,040	4,521	5,033	0,038	53,006
11	kvety	5,100	5,420	5,097	5,417	0,006	6,902
12	do	5,420	5,600	5,449	5,801	0,230	197,364
13	hory	5,600	5,840	6,382	6,430	1,372	401,197
14	peknú	6,410	6,780	6,446	6,718	0,098	184,364
15	kytičku	6,780	7,240	6,766	7,230	0,024	33,932
16	natrhal	7,240	7,900	7,310	7,838	0,132	299,842
17	rýchlo	7,900	8,330	7,934	8,478	0,182	122,222
18	lebo	8,560	8,820	8,542	8,814	0,024	11,405
19	mal	8,820	9,060	8,846	9,054	0,032	44,658
20	pre	9,120	9,270	9,086	9,294	0,058	15,816
21	kvety	9,310	9,690	9,326	9,662	0,044	13,696
22	cit	9,750	9,910	9,710	9,918	0,048	8,953
23	chcel	10,440	10,710	10,512	10,752	0,114	57,698
24	sa	10,710	10,930	10,768	10,976	0,104	67,750
25	presunúť	10,970	11,600	11,024	11,552	0,102	211,808
26	domov	11,600	12,030	11,632	12,064	0,066	50,669
27	čarovaním	12,030	12,820	12,144	12,784	0,150	121,035
28	ale	13,230	13,550	13,232	13,584	0,036	36,493
29	čo	13,610	13,720	13,616	13,776	0,062	60,349
30	sa	13,720	13,970	13,776	13,968	0,058	59,536

č.	**slovo**	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	Σ	ϕ
31	zrazu	13,970	14,310	14,000	14,384	0,104	124,405
32	stalo	14,310	14,920	14,480	15,248	0,498	275,997
33	baltík	15,520	16,030	15,526	16,006	0,030	33,692
34	nechtiac	16,030	16,650	16,086	16,550	0,156	164,158
35	namiesto	16,650	17,290	16,662	17,430	0,152	88,143
36	zaklínadla	17,560	18,490	17,606	18,582	0,138	88,332
37	nech	18,740	19,020	18,710	19,062	0,072	18,625
38	už	19,120	19,360	19,110	19,414	0,064	45,329
39	som	19,360	19,710	19,446	19,670	0,126	113,136
40	doma	19,710	20,050	19,702	20,054	0,012	10,662
41	povedal	20,490	20,980	20,509	21,133	0,172	95,149
42	nech	21,210	21,450	21,213	21,453	0,006	11,522
43	je	21,450	21,610	21,469	21,677	0,086	114,512
44	tu	21,610	21,810	21,693	21,869	0,142	173,405
45	najhorší	21,810	22,750	21,933	22,717	0,156	356,811
46	čarodejník	22,800	23,690	22,861	23,597	0,154	213,528
47	zrazu	23,870	24,230	23,886	24,302	0,088	150,507
48	sa	24,230	24,530	24,334	24,542	0,116	213,784
49	zjavil	24,530	24,930	24,574	24,910	0,064	112,830
50	ten	25,000	25,160	24,942	25,150	0,068	50,391
51	najhorší	25,160	25,930	25,230	25,902	0,098	195,544
52	čarodejník	25,970	26,670	26,014	26,622	0,092	285,887
53	sveta	26,670	27,180	26,718	27,166	0,062	67,146
54	vyzeral	27,480	27,990	27,493	27,941	0,062	75,769
55	hrozivo	27,990	28,650	28,005	28,405	0,260	505,566
56	z	28,910	29,030	28,453	28,661	0,826	583,181
57	očí	29,030	29,410	28,953	29,353	0,134	143,610
58	mu	29,410	29,560	29,385	29,465	0,120	210,271
59	šľahali	29,560	29,990	29,513	30,009	0,066	123,377
60	plamene	30,060	30,510	30,089	30,521	0,040	37,261
61	ohňa	30,580	31,000	30,585	30,985	0,020	6,997
62	mal	31,340	31,490	31,310	31,518	0,058	68,155

č.	slovo	Z_1	K_1	Z_2	K_2	Σ	ϕ
63	špicaté	31,490	32,120	31,582	32,078	0,134	344,516
64	nechty	32,120	32,590	32,158	32,574	0,054	80,987
65	vydával	32,970	33,460	33,000	33,512	0,082	56,830
66	strašidelné	33,460	34,250	33,624	34,264	0,178	148,732
67	zvuky	34,260	34,730	34,360	34,728	0,102	85,361
68	baltík	35,150	35,600	35,149	35,565	0,036	55,649
69	sa	35,530	35,800	35,597	35,789	0,078	82,276
70	zľakol	35,800	36,370	35,837	36,333	0,074	37,182
71	rýchlo	36,790	37,130	36,810	37,258	0,148	233,834
72	sa	37,130	37,280	37,290	37,370	0,250	517,039
73	začaroval	37,280	37,930	37,402	37,866	0,186	509,926
74	od	37,930	38,090	37,930	38,170	0,080	47,903
75	strachu	38,090	38,650	38,218	38,602	0,176	136,255
76	do	38,650	38,740	38,634	38,794	0,070	109,814
77	hrubého	38,740	39,500	38,842	39,434	0,168	261,866
78	hradného	39,500	40,100	39,530	40,106	0,036	60,789
79	múru	40,100	40,620	40,186	40,602	0,104	207,126
80	čarodejník	41,080	41,780	41,133	41,773	0,060	32,014
81	sa	41,780	42,000	41,837	41,981	0,076	61,689
82	od	42,000	42,150	41,981	42,157	0,026	12,843
83	zúrivosti	42,150	42,850	42,205	42,797	0,108	70,796
84	premenil	42,880	43,350	42,893	43,325	0,038	43,433
85	na	43,350	43,530	43,357	43,565	0,042	42,368
86	vodnú	43,550	44,000	43,613	44,013	0,076	71,842
87	paru	44,050	44,380	44,061	44,397	0,028	10,491
88	vznášal	44,890	45,270	44,887	45,287	0,020	24,281
89	sa	45,270	45,510	45,335	45,655	0,210	488,513
90	nad	45,510	45,700	45,703	46,135	0,628	1472,797
91	hradným	46,160	46,840	46,231	46,887	0,118	205,748
92	múrom	46,840	47,380	46,983	47,383	0,146	356,907
93	ako	47,650	47,960	47,541	47,973	0,122	42,288
94	ťažký	48,020	48,530	48,037	48,501	0,046	40,113

č.	**slovo**	$\mathbf{Z_1}$	$\mathbf{K_1}$	$\mathbf{Z_2}$	$\mathbf{K_2}$	$\mathbf{\Sigma}$	ϕ
95	oblak	48,570	49,100	48,581	49,045	0,066	19,896
96	baltík	49,520	49,930	49,525	49,925	0,010	5,490
97	sa	49,930	50,070	49,957	50,101	0,058	84,389
98	vypareného	50,070	51,130	50,181	51,061	0,180	299,980
99	čarodejníka	51,170	52,000	51,221	51,909	0,142	247,698
100	už	52,000	52,190	51,973	52,149	0,068	73,205
101	nebál	52,190	52,620	52,181	52,597	0,032	11,862
102	pokojne	52,990	53,470	53,005	53,469	0,016	3,153
103	išiel	53,470	53,900	53,533	53,869	0,094	103,995
104	domov	53,900	54,240	53,933	54,301	0,094	47,538
105	pešo	54,330	54,730	54,365	54,733	0,038	30,356
106	sľúbil	55,110	55,680	55,281	56,001	0,492	527,517
107	si	55,680	55,940	56,170	56,202	0,752	592,175
108	že	56,190	56,380	56,218	56,426	0,074	235,368
109	už	56,380	56,580	56,442	56,570	0,072	281,452
110	nikdy	56,580	56,910	56,586	56,922	0,018	13,913
111	sa	56,910	57,100	56,938	57,130	0,058	71,416
112	nebude	57,100	57,500	57,162	57,546	0,108	129,208
113	presúvať	57,590	58,220	57,626	58,186	0,070	66,352
114	domov	58,250	58,550	58,250	58,570	0,020	8,935
115	čarovaním	58,600	59,350	58,650	59,354	0,054	75,839
116	radšej	59,720	60,070	59,747	60,099	0,056	89,383
117	sa	60,070	60,250	60,147	60,355	0,182	217,879
118	prejde	60,350	60,800	60,403	60,867	0,120	82,267
119	aby	61,060	61,280	61,046	61,318	0,052	58,378
120	sa	61,280	61,490	61,334	61,510	0,074	98,908
121	vyhol	61,490	61,770	61,526	61,734	0,072	117,659
122	nebezpečenstvu	61,770	62,680	61,830	62,630	0,110	136,742
123	čarov	62,710	63,130	62,726	63,078	0,068	19,822

Dodatok B

Zoznam parametrov

Nasledujúca príloha je zhrnutím všetkých nastaviteľných parametrov vytvoreného algoritmu. Pre každý parameter uvádzam jeho názov, defaultnú hodnotu a krátky popis jeho významu. Podrobnejší popis jednotlivých parametrov sa nachádza v kapitole 4.

1. DLZKA_OKIENKA = 512

   Určuje použitú dĺžku mikrosegmentov (okienok).

2. PREKRYV_OKIENOK = 256

   Určuje, akou veľkou časťou sa budú mikrosegmenty prekrývať.

3. POLOMER_PREHLADAVANIA_SLOV = 0.5

   Určuje plus-mínus koľko slov skúsi algoritmus namapovať na každý blok okrem odhadnutého počtu.

4. MINIMALNY_POLOMER_PREHLADAVANIA_SLOV = 3

   Určuje minimálne plus-mínus koľko slov skúsi algoritmus namapovať na každý blok okrem odhadnutého počtu.

5. POLOMER_HLADANIA_ZACIATKU_BLOKU = 0.5

   Určuje veľkosť polomeru hľadania začiatočného slova bloku pri nesprávne určenom predchádzajúcom bloku.

6. MINIMALNY_POLOMER_HLADANIA_ZACIATKU_BLOKU = 5

   Určuje minimálnu veľkosť polomeru hľadania začiatočného slova bloku pri nesprávne určenom predchádzajúcom bloku.

7. GLOBALNE_OHRANICENIE_DTW = 20

   Určuje polomer prípustnej oblasti ohýbacej cesty.

8. HRANICNA_INTENZITA = 400

   Určuje hranicu intenzity pri rozdeľovaní signálu na bloky.

9. DLZKA_INTERVALU_INTENZITY_ROZSEKAVANIE = 0.3

   Určuje dĺžku intervalu počítania priemernej intenzity pri rozdeľovaní signálu na bloky.

10. DLZKA_INTERVALU_INTENZITY_OSEKAVANIE = 0.05

    Určuje dĺžku intervalu počítania priemernej intenzity pri určovaní tesných ohraničení blokov a slov.

11. OSEKANIE_IZOLOVANYCH = 1

    Určuje tesnosť ohraničenia pri určovaní tesných ohraničení izolovaných slov.

12. OSEKANIE_BLOKOV = 0.25

    Určuje tesnosť ohraničenia pri určovaní tesných ohraničení blokov súvislého signálu.

13. MIN_DLAZKA_IZOLOVANYCH = 0.1

    Určuje aké najkratšie slová po doladení hraníc sú prípustné.

14. MAXIMALNA_POVOLENA_DLZKA_BLOKU = 8.0

    Určuje maximálnu prípustnú dĺžku blokov po rozdelení signálu súvisle nahovoreného príbehu.

15. MINIMALNA_POVOLENA_DLZKA_BLOKU = 0.45

    Určuje minimálnu prípustnú dĺžku blokov po rozdelení signálu súvisle nahovoreného príbehu.

16. CESTA = c:\pokus\110\

    Určuje cestu pracovného adresára. Jej defaultnou hodnotou je prázdny reťazec.

17. SUVISLE = suvisle.mp3

    Určuje názov vstupného súboru - súvisle nahovoreného príbehu.

18. IZOLOVANE = izolovane.mp3

    Určuje názov vstupného súboru - po slovách nahovoreného príbehu.

19. IZOLOVANE_INDEXY = izolovane.mlab

    Určuje názov vstupného súboru - obsahujúceho začiatky a konce slov v súbore obsahujúcom po slovách nahovorený príbeh.

20. SUVISLE_INDEXY = suvisle.mlab

    Určuje názov výstupného súboru - obsahujúceho začiatky a konce slov v súbore obsahujúcom súvisle nahovorený príbeh.

21. TEST = suvisle-handwork.txt

    Určuje názov testovacieho súboru.

22. VAHA_CHYBY = 0.005

    Určuje veľkosť rastu chyby so vzdialenosťou od odhadnutého počtu slov.

23. DELTY = 0

    Určuje aký typ vektorov príznakov sa použije.

24. POCITANIE_VZDIALENOSTI = 0

    Určuje ako sa bude počítať vzdialenosť dvoch vektorov príznakov.

25. POSUN_STREDOV_GAUSIANOV = 0.25

    Určuje posun stredov dvoch nových gausiánov oproti pôvodnému z ktorého vznikli.

26. DOPASOVANIE_SLOV = 0.1

    Určuje, do akej vzdialenosti od koncov slova sa použije v závere slajdovacie DTW na nájdenie jeho presného začiatku a konca.

Dodatok C

Obsah priloženého CD

Na CD k diplomovej práci "Multimediálna čítanka"sa nachádza nástroj na transkripciu súvisle prečítaného príbehu s využitím transkripcie po slovách prečítaného toho istého príbehu.

Na správnu funkciu tohto nástroja je nutné mať nainštalovanú J2SE Runtime enviroment 5.0 Update 11. Môžete si ju stiahnuť z http://www.sun.com/

C.1 Obsah CD

CD obsahuje hlavný adresár *Dipl_MC* obsahujúci adresáre *Test_A* a *Test_B*, kde sa nachádzajú vstupné súbory dvoch testovacích príkladov, v adresári *zdrojaky* sa nachádzajú zdrojové kódy a v adresári *transkriptor* sa nachádza samotný nástroj.

C.2 Použitie nástroja

Pred spustením je nutné skopírovať súbory na pevný disk, pretože program zapisuje svoj výstup do súboru. Program je možné pustiť z príkazového riadku príkazom

```
java -jar -Xmx1000M MC.jar config.cfg
```

Parameter *-Xmx1000M* je nutný na zvýšenie maximálnej použitej operačnej pamäti. Pre dlhé príbehy bude možno nutné túto hodnotu ešte zväčšiť. Parameter

config.cfg určuje názov konfiguračného súboru, ktorý sa má použiť. Ak umiestnime konfiguračný súbor mimo adresára v ktorom sa nachádza súbor *MC.jar*, je nutné napísať celú cestu k nemu.

V adresári *transkriptor* sa nachádza aj dávkový súbor *MC.bat*, ktorý program spustí automaticky s použitím priloženého konfiguračného súboru.

V konfiguračnom súbore sú nastavené defaultné hodnoty parametrov a parameter *CESTA*, určujúci pracovný adresár, je nastavený na adresár *Test_A*.

Počas behu algoritmu sa vypisuje v akej fáze sa výpočet nachádza a informácie o vytváraných výstupných súboroch.