Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

DIPLOMOVÁ PRÁCE

PLZEŇ, 2016 **MICHAL KLÍMA**

PROHLÁŠENÍ

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí. Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

V Plzni dne

.....

PODĚKOVÁNÍ

Chtěl bych poděkovat Doc. Ing. Jindřichu Matouškovi, Ph.D., vedoucímu mé diplomové práce, za cenné rady a trpělivost. Dále bych chtěl poděkovat mé rodině a mojí přítelkyni Kačce, za neochvějnou podporu a dodávání tolik potřebné energie.

Abstrakt

Tato práce je zaměřena na vytvoření automatické fonetické segmentace s využitím nástroje Kaldi. Algoritmus segmentace je vytvořen na základě postupů použitých pro automatickou fonetickou segmentaci s pomocí nástroje HTK. Rozdíl v přesnosti obou segmentací se pohybuje pouze okolo 1%. Dále jsou zkoumány možnosti vylepšení automatické segmentace (nové modely, různé parametrizace, atd.). Z vyhodnocení výsledků vyplývá, že především pro nový model je přesnost segmentace o několik procent zvýšena.

Klíčová slova: HTK, Kaldi, Automatická fonetická segmentace, HMM, Syntéza řeči

Abstract

This work is focused on automatic phonetic segmentation using tool named Kaldi. Algorithm is developed based on the procedures used for automatic phonetic segmentation using tool named HTK. The difference in accuracy of both automatic segmentation is only around 1 %. There are also examined ways to improve automatic segmentation (new models, different parameterizations, etc.). The evaluation indicate that especially for the new model is a segmentation accuracy of a few percent increased.

Keywords: HTK, Kaldi, Automatic phonetic segmentation, HMM, Speech synthesis

Obsah

Kapitola 1

Úvod

Zpracování řeči je velice široká oblast, která je ve středu zájmu mnoha vědeckých i komerčních sfér. Úlohy zpracování řeči mohou být rozděleny podle nejrůznějších faktorů. Mezi jedny z nejširších oborů můžeme zařadit rozpoznávání řeči a syntézu řeči. Právě do oblasti syntézy řeči, neboli strojového vytváření umělé řeči, se snaží přispět tato diplomová práce.

Syntéza řeči je vyvíjena už poměrně dlouhou dobu a proto se v této práci zmiňujeme i o méně používaných a mnohdy už historických metodách vytváření umělé řeči. Hlavní část pozornosti je však upřena na moderní metody a některé postupy s nimi spojené, které zde budeme postupně popisovat. Za jeden z nejpoužívanějších přístupů k umělému vytváření řeči můžeme označit *konkatenační syntézu řeči*. Této metodě se detailněji věnujeme v části 2.2, ale zjednodušeně lze říci, že tato metoda vytváří umělou řeč spojováním určitých segmentů původní řeči namluvené řečníkem. Z uvedeného se zdá být logické, že pro vytváření umělé řeči touto metodou je důležité znát přesné hranice zvolených segmentů původní řeči, neboli přesný čas jejich začátku a konce. Proces určování těchto hranic se nazývá *segmentace řeči*, které se věnujeme především v části 3. Dříve se segmentace prováděla ručně, což byla ovšem velmi náročná a zdlouhavá práce. Navíc intenzivní vývoj v této oblasti a s ním spojený nárůst objemu řečových dat učinil tento způsob prakticky neproveditelný. Začaly se tak vytvářet automatické algoritmy segmentace, což je hlavní téma této práce.

Tato diplomová práce vznikla v návaznosti na předchozí výzkum v této oblasti na Katedře kybernetiky (na Západočeské univerzitě v Plzni). V tomto výzkumu byla automatická segmentace vytvořena s pomocí nástroje HTK, kterému je věnována část 4.1. Tento nástroj je však v současné době považován za zastaralý, s malou podporou a téměř nulovým vývojem. Je tak žádoucí hledat nové nástroje, které budou schopné provádět automatickou segmentaci stejně dobře jako HTK, nebo i lépe. Jedním z těchto nástrojů je nástroj Kaldi, kterému je věnována část 4.2. Kaldi je v současné době intenzivně vyvíjeno a podporuje, jak zaběhnuté a funkční metody ze starších nástrojů, tak i nové metody. Nový algoritmus automatické segmentace řeči, kterým se tato práce zabývá, je tak vytvořen právě s pomocí nástroje Kaldi.

Dalším tématem, kterým se tato práce zabývá, je implementace několika modifikací automatické segmentace, ve snaze zvýšení přesnosti segmentace. Jsou zde například pro-

zkoumány možnosti nových modelů pro modelování hlásek a pauz, nebo použití různých řečových parametrizací. Všechny modifikace automatické segmentace jsou popisovány v části 3 a především v části 5.

Poslední částí této práce je vyhodnocení dosažených výsledků automatické segmentace s využitím nástroje Kaldi, jak oproti předešlému algoritmu (s využitím nástroje HTK), tak vzhledem ke zmíněným modifikacím. Vyhodnocení se věnuje část 6.

Kapitola 2

Syntéza řeči

Syntézou řeči zpravidla rozumíme vytváření umělé řeči, které provádí syntetizér řeči. Ten má na vstupu informaci o tom, jaká řeč se má generovat a jakým způsobem, a na jeho výstupu se pak vytvoří umělá řeč. Vstupní informací bývá fonetická a prozodická informace o generované řeči. Fonetickou informací je posloupnost hlásek, tedy popis jaká řeč se má vytvořit. Prozodickou informací jsou průběhy základních prozodických charakteristik (například melodie, časování, atd.), tedy popis, jak se má řeč vytvořit. Hlavními požadavky na syntetizér jsou především srozumitelnost a přirozenost výsledné řeči. Ideálním výsledkem by tedy bylo, aby byl syntetizér schopný vytvořit takovou umělou řeč, která bude k nerozeznání od řeči vyřčené člověkem.

Původně se přístupy k syntéze řeči rozdělovaly do třech základních skupin, které se lišily způsobem modelování použitém při vytváření umělé řeči. S vývojem v této oblasti se ovšem toto rozdělení změnilo. V části 2.1 jsou popsány původní metody, které se dnes už téměř nepoužívají. Vzhledem k současnému trendu v přístupu k syntéze řeči a k povaze této práce se těmito přístupy zabýváme pouze okrajově (více o nich lze nalézt například v [19]). Hlavní přístup k syntéze řeči, který se v dnešní době používá, je *korpusově založená syntéza řeči*. Tu lze rozdělit na část *konkatenační syntézy* (více v části 2.2) (často s využitím metody *unit selection* (více v části 2.6) a *statistickou parametrickou syntézu řeči* (více v části 2.5).

2.1 Starší metody syntézy řeči

V této části si popíšeme některé starší metody syntézy řeči, které se však dnes už používají minimálně. Mezi tyto metody patří *formantová syntéza řeči* a *artikulační syntéza řeči*.

2.1.1 Artikulační syntéza řeči

Artikulační syntéza řeči se snaží přesně napodobit vytváření řeči člověkem. Modeluje tedy celý proces vytváření řeči, od vytvoření zdroje energie (vzduchu), jeho pohybu skrz hlasové ústrojí člověka, až po dokončení výsledné řeči v ústní dutině. Tento přístup je však velmi složitý a syntetizér, který by vytvářel srozumitelnou a zároveň přirozenou řeč, na takové

úrovni, aby byl použitelný v komerčních produktech, ještě ani nebyl vytvořen.

2.1.2 Formantová syntéza řeči

Přístup formantové syntézy řeči je založen na teorii zdroje a filtru. Zdroj zde představuje zdroj buzení, který generuje posloupnost hlasivkových pulsů pro znělé zvuky a náhodný šum pro zvuky neznělé. Hlasový trakt zde představuje filtr, jehož parametry jsou spjaty zejména s formanty[1] hlasového traktu . Formantová analýza pro vytváření umělé řeči dále používá soubor pravidel vytvořených expertem (např. frekvenci a šířku pásem formantů), které se vysokou měrou podílejí na kvalitě výsledné řeči.

2.2 Konkatenační syntéza řeči

Konkatenační syntéza řeči v současnosti patří k nejvíce rozvíjeným způsobům vytváření umělé řeči. Základní idea se opírá o skutečnost, že jsme schopni prakticky jakékoli umělé zvuky vytvářet za pomoci konečného počtu řečových jednotek. Řečovou jednotku můžeme chápat jako abstraktní pojem pro pojmenování stejného typu řečových zvuků, například fonémů (viz kapitola 2.4). Konkatenační syntézou poté rozumíme konkatenaci, neboli zřetězení, těchto řečových jednotek za účelem vytvoření řečového signálu. Řečové jednotky jsou uloženy v inventáři řečových jednotek, kterým se zpravidla myslí označkovaný, neboli nasegmentovaný korpus (více v kapitole 2.3), ve kterém byli nalezeny hranice řečových jednotek.

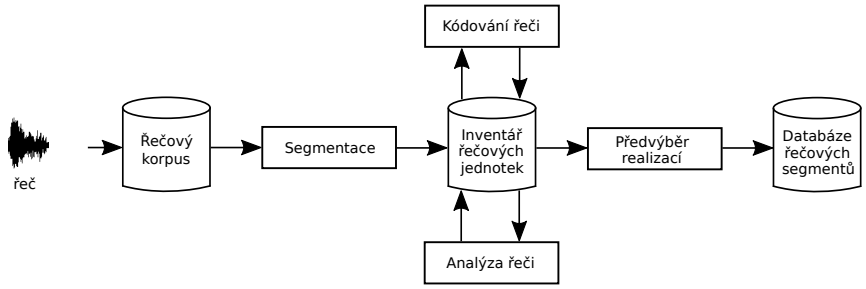
Základní rozdělení konkatenační syntézy tak můžeme provést podle způsobu získávání řečových jednotek na *ruční* (hranice řečových jednotek v korpusu jsou hledány expertem) a *automatické* (hranice řečových jednotek v řečovém korpusu jsou hledány automaticky), zde hovoříme o korpusově orientované syntéze řeči (více v kapitolách 2.5 a 2.6). Další rozdělení konkatenační syntézy, lze provést podle způsobu reprezentace řečových segmentů. Pokud pracujeme se segmenty v časové oblasti, tedy přímo se vzorky řečového signálu, mluvíme o *neparametrické reprezentaci*. Pokud pracujeme s parametry řečového signálu, tedy se segmenty zakódovanými pomocí frekvenčních parametrů (MFCC, PLP - více v kapitole 2.7), hovoříme o *parametrické reprezentaci* a syntéza probíhá ve frekvenční oblasti. Posledním důležitým rozdělením, které si uvedeme je rozdělení konkatenační syntézy podle velikosti slovníku. První možností je *syntéza s omezeným slovníkem*, která pracuje pouze v omezené oblasti (například hlášení příjezdů a odjezdů vlaků na nádraží) a zpravidla nebývá schopná generovat libovolnou řeč. Druhou možností je *syntéza s neomezeným slovníkem* používaná v systémech konverze textu na řeč (TTS). Používají se především menší jednotky (fonémy, difony, trifony, atd.). Tato práce spadá do oblasti *konkatenační syntézy s automatickým vytvářením řečových jednotek a neomezeným slovníkem.*

Hlavní výhodou konkatenační syntézy je vysoká kvalita výsledné generované řeči, a to díky práci s reálnými úseky skutečné řeči namluvené řečníkem. Tento přístup ovšem přináší i určité nevýhody. Konkatenací řečových segmentů, které spolu v původní řeči

[1] Formantem rozumíme místo koncentrace akustické energie.

nesousedily, nastává problém právě v místech jejich spojení. Například vlivem koartikulace se ve výsledné řeči mohou objevit jisté spektrální, či prozodické nespojitosti. Řešení těchto problémů se lze dozvědět například v [19].

Na obrázku 2.1 můžeme vidět základní schéma konkatenační syntézy, jejíž postup si nyní přiblížíme. Prvním krokem je volba řečových jednotek a zvolení způsobu vytvoření jejich inventáře. Dalším krokem je vytvoření řečového korpusu, tak aby v něm každá zvolená jednotka byla alespoň jednou zastoupena. Korpus je poté segmentován, aby byly zjištěny hranice jednotlivých řečových jednotek, čímž vznikne segmentovaný řečový korpus. Jednotlivé segmenty jsou, společně s dalšími informacemi uloženy do inventáře řečových jednotek. Dále jsou, podle informace o generované promluvě na vstupu syntetizéru vybrány vhodné řečové jednotky (v inventáři může být více reprezentantů jedné řečové jednotky). Jednotlivé segmenty je pak vhodné dodatečně modifikovat, jelikož většinou vykazují různé prozodické vlastnosti (byly získány z odlišného kontextu, než do jakého jsme je nyní zasadili). Poslední fází je již samotné zřetězení segmentů, které je ještě doplněno o vyhlazování akustických přechodů, aby bylo zabráněno projevům změn spektrálních charakteristik. Více informací se lze dozvědět například v [19], nebo v [23]. Vzhledem k povaze práce se budeme detailněji zabývat pouze částí **segmentace řečových jednotek** (viz kapitola 3).



Obrázek 2.1: Princip konkatenační analýzy (převzato z [19]).

2.3 Řečové korpusy

Jako řečový korpus můžeme označit soubor libovolných řečových záznamů obohacených o dodatečné informace, například anotaci a transkripci promluv. Existuje mnoho rozdělení korpusů, ať už z hlediska zastoupení jednotlivých řečových jednotek (bohatý korpus, vyvážený korpus), počtu řečníků, kteří celý korpus namlouvali, nebo podle prostředí ve kterém byl daný korpus nahráván.

Při nahrávání korpusu se nejprve řeší jeho účel, podle kterého se následně zvolí jeho velikost, počet řečníků, míra zastoupení jednotlivých řečových jednotek a prostředí nahrávání. Další částí je samotné nahrávání řečového záznamu, který je v poslední fázi anotován (označení způsobu vyslovení promluvy - šumy, neřečové události, atd.) a je vytvořena transkripce všech promluv (přesný přepis toho, co bylo v promluvě řečeno). Právě na základě

anotace řečového korpusu je možné vytvořit slovník všech slov, který poté hraje důležitou roli, jak při samotné syntéze řeči, tak při automatické segmentaci, kterou se zde budeme zabývat. Více o řečových korpusech lze zjistit například v [19].

2.4 Řečové jednotky

Řečovou jednotkou můžeme nazvat prakticky libovolný úsek řeči. Pokud bychom si chronologicky (od nejmenších až po největší) uspořádali alespoň některé z nich, začali bychom pravděpodobně na hláskách, nebo fonémech a pokračovali přes difony, trifony, slabiky, až po slova a celé fráze. Volba řečové jednotky, která bude použita, závisí na mnoha faktorech. Jedná se především o její bezproblémové řetězení, maximální pokrytí koartikulačních jevů, zobecnitelnost a v neposlední řadě by zvolených řečových jednotek nemělo být v daném jazyku příliš velké množství. Počet řečových jednotek v daném jazyku totiž úzce souvisí s následnou velikostí korpusu a také mnohem většími nároky na úložné prostory počítače. Pokud řešíme úlohy s velmi omezeným slovníkem je obecně výhodnější použít delší řečové jednotky (slova, fráze), protože výsledek bude znít přirozeně, jelikož se sníží počet konkatenačních bodů. S rostoucí velikostí slovníku se získávání a ukládání takto velkých jednotek stává velice náročným a při použití neomezeného slovníku už takřka nemožným. Ve většině úloh syntézy řeči se tedy používají menší jednotky a to především fonémy a trifony. Více informací lze nalézt například v [6], nebo v [19].

2.4.1 Fonémy

Představují nejmenší přirozené jednotky řeči, schopné rozlišovat významové jednotky. Jako dva různé fonémy můžeme uvést například hlásky „a“ a „á“, protože odlišují význam slov například „pas“ a „pás“. Množina fonémů je různá pro každý jazyk, ovšem vždy jich je poměrně malé množství (například v češtině je 40 fonémů) a na první pohled se tak zdají vhodnými kandidáty na kvalitní řečovou jednotku. Fonémy však nepostihují okolní kontext a jelikož se výslovnost fonémů mění právě v závislosti na okolním kontextu není jejich samostatné použití dostačující. Nabízí se tak možnost použití řečových jednotek, které jsou kontextově závislé. Nejpoužívanějšími kontextově závislými řečovými jednotkami jsou *difony* a *trifony*.

2.4.2 Trifony

Trifony jsou řečové jednotky, které postihují bezprostřední pravý i levý kontext, nejedná se tedy o tři hlásky za sebou, jak by se mohlo z názvu zdát. Jako příklad takového trifonu můžeme uvést – n-o+c , kde jde o trifon o s bezprostředním levým kontextem n, značeným pomocí znaku „-“ a bezprostředním pravým kontextem c, značeným pomocí znaku „+“. Definice trifonu se může různit, ale standardně o něm můžeme mluvit jako o kontextově závislé hlásce. Pokud předpokládáme, že máme N hlásek, tak teoretický počet trifonů bude N^3 a uchovávat všechny jejich realizace by bylo neúnosné. Pro zredukování jejich počtu se

používají algoritmy shlukování, které trifony, odvozené od stejné hlásky shluknou do tříd s podobnými vlastnostmi. Pro shlukování je možné použít standardní MacQueenův algoritmus, ale častěji se využívá klasifikačních a regresních stromů. Více se této problematice věnujeme v části 3.4.3.

2.4.3 Difony

Difony nejsou kontextově závislou řečovou jednotkou ve stejném významu jako například trifon definovaný v předchozí části. Jejich délka odpovídá přibližně délce jedné hlásky. Rozdíl od klasické hlásky je ovšem v tom, že difon začíná přibližně v polovině hlásky a končí přibližně v polovině hlásky následující. Důvod zavedení difonu vychází především z předpokladu, že uprostřed dané hlásky je vliv kontextu okolních hlásek nejmenší. Postupně se však zjistilo, že okolní kontext hlásky může mít vliv i na její celou délku a tak se difonové inventáře začali doplňovat vhodně zvolenými delšími řečovými jednotkami (například trifony). Jelikož je difonů relativně malé množství (při N hláskách existuje N^2 difonů a skutečně použitelných je ještě méně), nacházejí i dnes využití v konkatenačních syntetizérech.

2.5 Statistická parametrická syntéza řeči (HMM syntéza)

Statistické parametrická syntéza řeči nepracuje přímo s jednotlivými segmenty řeči na signálové úrovni, ale spíše s jejich statistickými modely. Pro tyto účely se výhradně využívá skrytých Markovových modelů (z anglického spojení Hidden Markov Models (HMM), proto HMM syntéza), kterým se detailněji věnujeme v části 3.2. Tato technika ke své práci využívá obrovské korpusy (v řádech několika desítek hodin řeči) a její postup může být rozdělen do dvou částí – trénovací fáze a fáze syntézy.

V trénovací části jsou trénovány modely tak, aby byla maximalizována pravděpodobnost generování sledovaných parametrů těmito modely. Ve fázi syntézy se na základě vstupních promluv, reprezentovaných posloupností fonémů, zřetězí odpovídající HMM a výsledná řeč je generována zdrojem buzení na základě sekvence získaných řečových parametrů. Více o této metodě se lze dozvědět v [24] a [11].

2.6 Konkatenační syntéza s výběrem jednotek (Unit selection)

Tato metoda se řadí mezi nejvýznamnější metody konkatenační syntézy řeči. Na základě různých pravidel a hodnotících funkcí jsou vybrány nejvhodnější realizace řečových jednotek z inventáře, které se následně řetězí. Kvalita výsledné řeči zde úzce souvisí s kvalitou řečového korpusu a tak se často používají velké řečové korpusy, které obsahují každou řečovou jednotku v různých fonetických, spektrálních a prozodických realizacích.

Syntéza s výběrem jednotek má na vstupu informaci o požadované řeči, kterou chceme syntetizovat. Vedle posloupnosti jednotek to jsou další, například prozodické, informace. Úloha syntézy s výběrem jednotek poté spočívá v nalezení optimální posloupnosti řečových jednotek v nasegmentovaném korpusu. Jednotky jsou řetězeny a následně vyhlazovány, přičemž se předpokládá, že čím přesnější posloupnost jednotek v řečovém korpusu najdeme, tím méně úprav výsledné syntetické řeči budeme muset provádět.

Jak už bylo zmíněno, při hledání optimální posloupnosti řečových jednotek se používají hodnoticí funkce. Tyto hodnoticí funkce jsou dvě a to *cena cíle (target cost)*, která popisuje jak se konkrétní instance řečové jednotky liší od požadované jednotky a *cena zřetězení (join cost)*, která vyjadřuje jak dobře se dvě instance řečových jednotek řetězí. Optimální posloupnost lze najít sečtením těchto dvou hodnotících funkcí pro všechny posloupnosti řečových jednotek:

$$C(U,S) = \sum_{t=1}^{N} T(u_t, s_t) + \sum_{t=1}^{N-1} J(u_t, u_{t+1}), \tag{2.1}$$

kde U je posloupnost řečových jednotek, S je vektor příznaků (skládá se ze spektrálních koeficientů, lokální spojitosti energie, pozičních parametrů, atd.), T(.) je cena cíle a J(.) je cena zřetězení. Pro všechny možné posloupnosti řečových jednotek se tento vzorec vypočítá a optimální posloupnost řečových jednotek je vybrána podle:

$$\hat{U} = \underset{u}{\operatorname{argmin}} \left\{ C(U,S) \right\}. \tag{2.2}$$

Vhodným algoritmem pro nalezení optimální posloupnosti je *Viterbiův algoritmus*. Více informací o syntéze s výběrem jednotek lze získat například v [23], [19], nebo v [11].

2.7 Analýza řeči

2.7.1 Melovské kepstrální koeficienty (MFCC)

Přístup pro získání MFCC koeficientů vychází z myšlenky rozdělit řečový signál na mikrosegmenty o určité délce. Na takto vytvořené časové úseky se aplikují okénka (nejčastěji Hammingovo), aby je bylo možno zpracovat pomocí rychlé Fourierovy transformace (FFT). Okénko se postupně posouvá o pevně stanovený krok, a v každém takovém kroku je prováděna melovská filtrace. Tato filtrace spočívá v použití banky trojúhelníkových pásmových filtrů s lineárním rozložením v melovské frekvenční škále. Po průchodu signálu těmito filtry je tedy každý koeficient FFT násoben odpovídajícím ziskem filtru a tyto výsledky jsou akumulovány. Posledním krokem je logaritmování všech jednotlivých výstupů filtru

s následným provedením zpětné diskrétní Fourierovy transformace, čímž získáme výsledné MFCC:

$$c_m(j) = \sum_{i=1}^{M^*} log\ y_m(i) cos(\frac{\pi j}{M^*}(i - 0,5)), \quad pro\ j = 0, 1, ..., M, \tag{2.3}$$

M^*je počet pásem melovského pásmového filtru, M je počet melovských kepstrálních koeficientů a y_m jsou výstupy jednotlivých filtrů. Více informací lze získat například v [19]

Postup, který jsme použili pro výpočet MFCC parametrů má jednu zásadní nevýhodu. MFCC parametry získané touto cestou jsou pouze statické. K těmto standardním koeficientům se tedy často doplňují dodatečné příznaky označované jako delta a delta-delta koeficienty, souhrnně nazývané dynamické koeficienty. Ty nám jsou schopny vyjádřit dynamiku časové změny vektorů příznaků, pro každý analyzovaný mikrosegment, za pomoci lineární regrese po sobě jdoucích mikrosegmentů. Využívají se především pro svou nekorelovanost a vysoký informační obsah. Výsledný vektor parametrizací poté sestává z $M + 1$ statických kepstrálních Melovských koeficientů, $M + 1$ delta koeficientů a $M + 1$ delta-delta koeficientů:

$$c_m^{all} = [c_m, \triangle c_m, \triangle^2 c_m]. \tag{2.4}$$

2.7.2 Koeficienty perceptivní lineární prediktivní analýzy (PLP)

Metoda PLP se snaží o zmírnění nedostatků běžných metod analýzy řeči, protože jejich popis spektrálních vlastností řečového signálu příliš neodpovídá způsobu, jakým zvuky vnímá člověk. Transformace výkonového spektra řečového signálu do odpovídajícího sluchového spektra je zde provedena kombinací kritického pásma spektrální citlivosti, křivky stejné hlasitosti a vztahu vyjadřujícím závislost mezi intenzitou a jeho vnímanou hlasitostí. Jednotlivé kroky PLP analýzy se dají rozdělit následovně:

- Výpočet výkonového spektra řečového signálu,
- Nelineární transformace frekvencí a kritická pásma spektrální citlivosti slyšení,
- Přizpůsobení kritických pásmových filtrů křivkám stejné hlasitosti,
- Vážená spektrální sumarizace vzorků výkonového spektra,
- Uplatnění vztahu vyjadřujícího závislost mezi intenzitou zvuku a vnímanou hlasitostí
- Aproximace spektrem celopólového modelu

Po sérii těchto operací je získán předpis pro výpočet autokorelační funkce:

$$R(i) = \frac{1}{2(M-1)} \left\{ \xi(\Omega_0) cos(i\omega_0) + 2 \left[\sum_{m=1}^{M-2} \xi(\Omega_m) cos(i\omega_m) \right] + \right. \\ \left. + \xi(\Omega_{M-1}) cos(i\omega_{M-1}) \right\}, \tag{2.5}$$

kde $i = 0, \ldots, Q$, $\omega_{M-1} = \pi$ a $\xi(\Omega_m)$ je aproximace zákona slyšení vyjadřujícího vztah mezi intenzitou zvuku a jeho vnímanou hlasitostí pro *m-tý* pásmový filtr ($m = 0, \ldots, M-1$). Poté lze hledané koeficienty lineární predikce nalézt například Durbinovým algoritmem, jako hodnoty autokorelační funkce $R(i)$ pro $i = 0, \ldots, Q$, kde Q je řád prediktoru PLP analýzy. Definice PLP parametrizace zde je velmi stručná, detailnější informace lze získat například v [19].

Kapitola 3

Segmentace řeči

Segmentací řeči rozumíme proces hledání hranic řečových jednotek v řečových promluvách, které jsou uložené v řečovém korpusu. Dříve bylo běžné provádět *ruční segmentaci*, zpravidla zkušeným expertem v dané oblasti. Byla to ovšem velmi frustrující a časově náročná práce, určené hranice řečových jednotek od jednotlivých expertů byly navíc často nekonzistentní a vznikali tak nepřesnosti v důsledku působení lidského faktoru. Řečové korpusy se navíc v důsledku intenzivního vývoje v oblasti konkatenační syntézy řeči stále zvětšovaly a tak se ruční segmentace pomalu dostala do pozadí zájmu. Pozornost se poté zaměřila na vytváření algoritmů pro *automatickou segmentaci*.

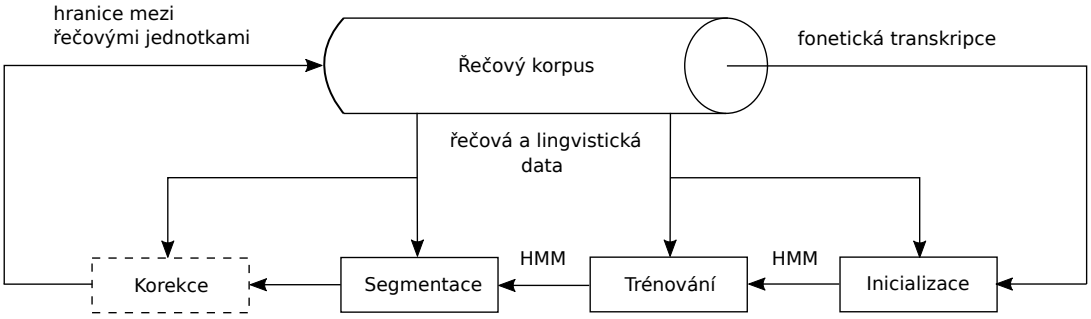
Jedním ze dvou nejčastěji využívaných principů, jak v dnešní době provádět automatickou segmentaci, je *využití techniky dynamického borcení času*. Tato metoda porovnává řečový signál, ve kterém známe hranice řečových jednotek se signálem, ve kterém je chceme získat. K tomuto účelu se využívá algoritmu dynamického borcení časové osy. Více informací o tomto algoritmu lze nalézt například v [19].

Zde se ovšem zaměříme především na druhý způsob, který využijeme v této práci a tím je *automatická segmentace s využitím mnohočetných skrytých Markovových modelů*. Obecně se nedá říci, že by tato technika byla lepší, než technika DTW, ale podle [11] není tak náchylná k vytváření velkých segmentačních chyb a tak je hlavně v případě konkatenační syntézy řeči její použití výhodnější. Zjednodušené schéma tohoto způsobu automatické segmentace můžeme vidět na obrázku 3.1.

Nejprve se zaměříme na definici skrytých Markovových modelů (dále jen HMM) (část 3.2), poté si ukážeme některé jejich topologie, které v této práci budeme využívat (část 3.3). Nakonec si popíšeme způsob trénování těchto modelů (část 3.4) a samotnou fázi segmentace (část 3.5).

3.1 Markovovy modely

Jak se lze dozvědět v [7], využití HMM vychází z problematiky rozpoznávání řeči, kde bylo třeba vyřešit především tři důležité problémy. Model musí být dostatečně flexibilní, což znamená, že rozpoznávač řeči musí být použitelný i za odlišných podmínek, než za



Obrázek 3.1: Zjednodušený proces automatické segmentace (převzato z [11]).

jakých byl natrénován. Model musí být přesný, aby bylo možné dostatečně dobře odlišit foneticky podobná slova s odlišnými lingvistickými vlastnostmi a model musí být použitelný v reálném čase. Všechny tyto aspekty splňují HMM. Při definici HMM je ovšem vhodnější nejprve definovat Markovův řetězec.

3.1.1 Markovův řetězec

Markovův řetězec můžeme chápat jako speciální verzi váženého konečného automatu, kde vstupní sekvence symbolů určí, kterými svými stavy automat při jejím zpracování projde. Markovův řetězec je specifikován:

- množinou svých stavů
$$Q = \{q_1, q_2, ..., q_N\} \tag{3.1}$$
- přechodovou maticí přechodů mezi jednotlivými stavy
$$A \ = [a_{01}, a_{02}, ..., a_{nn}] \tag{3.2}$$
- počátečním stavem
$$q_0 \tag{3.3}$$
- konečným stavem
$$q_F \tag{3.4}$$

Zajímavou vlastností Markovova řetězce, což je i důvod proč je tak široce používán je, že pravděpodobnost následujícího stavu závisí pouze na aktuálním stavu:

$$P(q_i|q_1...q_{i-1}) = P(q_i|q_{i-1}), \tag{3.5}$$

a součet všech pravděpodobností přechodu ze stavu i do ostatních stavů musí být rovný jedné:

$$\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i. \tag{3.6}$$

Více informací o Markovových řetězcích je možné dohledat v [6].

3.2 Skryté Markovovy modely (HMM)

Zatímco Markovovým řetězcem se velmi dobře modelují události, které jsou pozorovatelné, v mnohých případech musíme pracovat s událostmi, které přímo pozorovatelné nejsou. Jako příklad můžeme uvést řečový signál, kde vidíme, respektive slyšíme, slova, která jsou na vstupu, ale už nemůžeme pozorovat jednotlivé stavy artikulačního orgánu. Právě pro tyto účely se hodí modelování za pomoci HMM. HMM je automat, který generuje vektor pozorování v diskrétních časových okamžicích. Podle [24] lze definici HMM zavést takto:

N-stavový HMM je definován přechodovou maticí přechodů mezi jednotlivými stavy $A = \{a_{ij}\}_{i,j=1}^{N}$, hustotní funkcí výstupních vektorů pozorování $B = \{b_i(o)\}_{i=1}^{N}$ a počáteční inicializací pravděpodobnosti stavů $\pi = \{\pi_i\}_{i=1}^{N}$

Zjednodušeně lze tedy model HMM zapsat následující trojicí parametrů :

$$\lambda = (A, B, \pi) \quad . \tag{3.7}$$

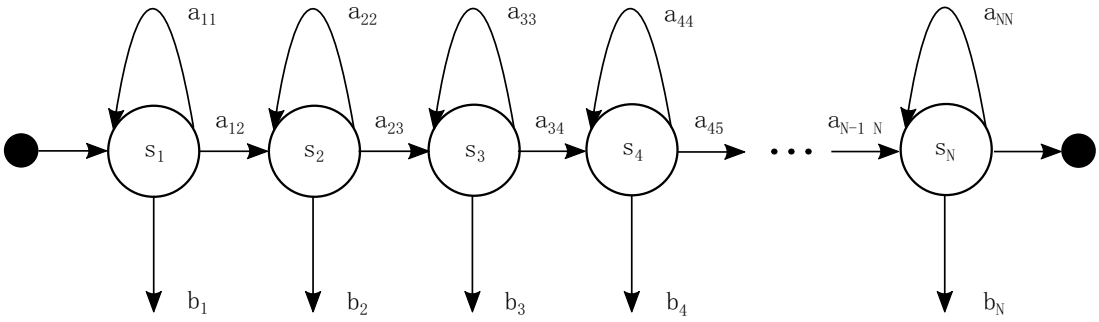
Rozdělení výstupní pravděpodobnosti musí být variabilní, ale zároveň musí být také dostatečně robustní. Mezi nejčastěji užívaná rozdělení patří *spojité rozdělení se směsí normálních hustotních funkcí*, *spojité rozdělení se svázanou směsí normálních hustotních funkcí* a *diskrétní rozdělení*. Více informací o uvedených typech rozdělení lze nalézt v [19].

3.3 Topologie HMM

Topologie HMM, potažmo její úprava, je jednou ze stěžejních částí této práce, takže se teď na ní zaměříme detailněji. Topologii HMM je možné volit prakticky libovolně. Nejčastěji používanými modely při modelování mluvené řeči jsou *levo-pravé Markovovy modely*. Struktura těchto modelů vychází z přirozeného pohledu na vlastnosti řečového signálu. Jednotlivé stavy modelu představují určité nastavení artikulačního ústrojí při vyslovování daného úseku řeči (celého slova, hlásky, slabiky, atd.) a přechody mezi jednotlivými stavy jsou pouze dopředné (odtud název levo-pravé), stejně jako se řeč nevrací v čase.

Jedním z prvních takových modelů byl tzv. Bakisův model, který se používal především pro modelování celých slov. Obsahoval přibližně 40-60 stavů, při mikrosegmentu o délce 10 ms a umožňoval přechody pro dloužení i krácení slov. Jeho grafické znázornění můžeme vidět na obrázku 3.2. Postupem času se však zjistilo, že se tyto modely k automatické segmentaci nehodí, protože bylo velmi složité najít vztahy mezi jednotlivými stavy HMM a lingvistickou reprezentací, které potřebujeme při automatické segmentaci znát.

Začala se tak vytvářet řada dalších modelů, které z těchto původních vycházely a snažily se je nějakým způsobem vylepšit. Šlo především o redukci počtu stavů, protože s vývojem



Obrázek 3.2: Bakisův model slova.

v oblasti syntézy a rozpoznávání řeči bylo zapotřebí modelovat menší úseky řeči, než slova, například hlásky. Bakisův model byl pro modelování takových řečových jednotek zbytečně složitý a při jeho použití zůstávala většina jeho stavů nakonec nevyužita. Pozornost se tak přesunula k *mnohočetným HMM*, kde každému modelu odpovídá jistá lingvistická značka[1]. Tyto modely se dnes používají pro modelování, jak *kontextově nezávislých* řečových jednotek (označovaných jako monofony – fonémy, hlásky, atd.), tak i *kontextově závislých* (trifony).

Po mnoha experimentech s různými modely se stal nejpoužívanějším 5-ti stavový levopravý HMM se třemi emitujícími[2] stavy, který můžeme vidět na obrázku 3.3. Tento model se ukázal jako nejvhodnější vzhledem ke své jednoduchosti, robustnosti a jeho stavy zároveň dobře reprezentují určité (kvazi)stacionární části hlásek.

Ačkoliv tento model rozhodně můžeme považovat za nejpoužívanější, při modelování hlásek, existuje mnoho dalších modelů, které se používají pro modelování některých speciálních řečových jednotek (například pauz).

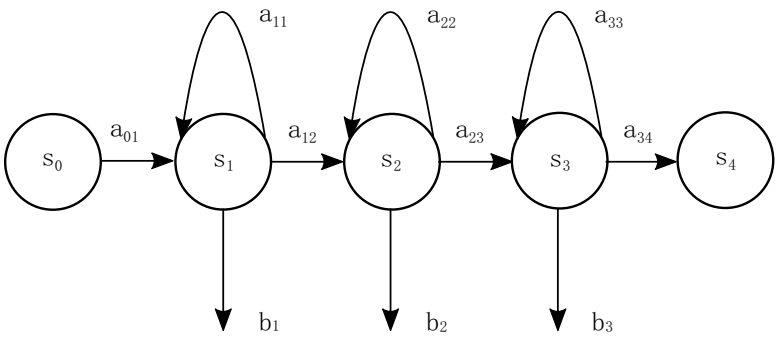
Upravené modely pauz patří k široce používaným a známým. Jedná se především o model dlouhé pauzy (běžně označované jako *SIL*, viz obrázek 3.4a) a model krátké pauzy (běžně označované jako *SP*, viz obrázek 3.4b). Dlouhá pauza se zpravidla používá na začátku a konci věty. Krátká pauza se většinou používá pro modelování pauzy mezi slovy, přičemž její topologie dovoluje model kompletně „přeskočit“.

Nyní se zaměříme na možnost vylepšení 5-ti stavového modelu hlásky, který lze vidět na obrázku 3.3. V [20] je diskutována možnost rozšíření krajních stavů modelu o dodatečné stavy, což můžeme vidět na obrázku 3.5. Dodatečné stavy jsou vytvořeny jako kopie krajních stavů původního, nerozšířeného modelu a self-loop přechody (přechody ze stavu do téhož stavu) jsou u všech těchto kopií nahrazeny přechodem do prvního, respektive posledního stavu. Takto modifikovaná[3] topologie HMM by tedy měla vylepšit kontrolu

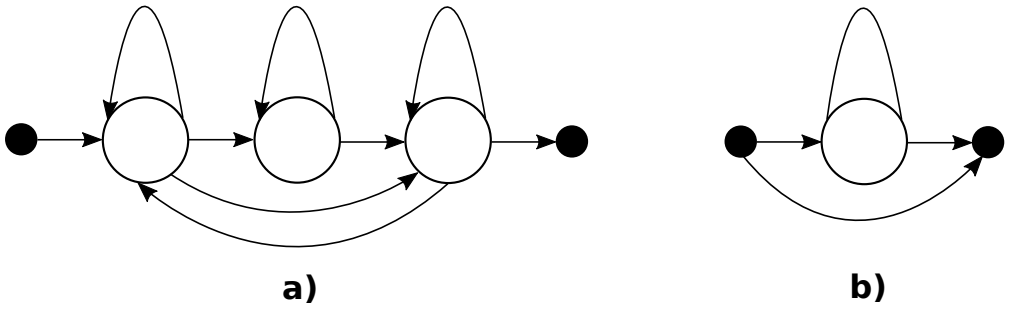
[1]Ta odpovídá zvolené řečové jednotce. Může jí tedy být například hláska, foném, atd.

[2]Stavy schopné generovat výstupní vektor pozorování.

[3]Pokud se v této práci bude mluvit o modifikované topologii HMM, bude tím vždy myšlena topologie z obrázku 3.5.



Obrázek 3.3: 5-ti stavový levo-pravý HMM se třemi emitujícími stavy.

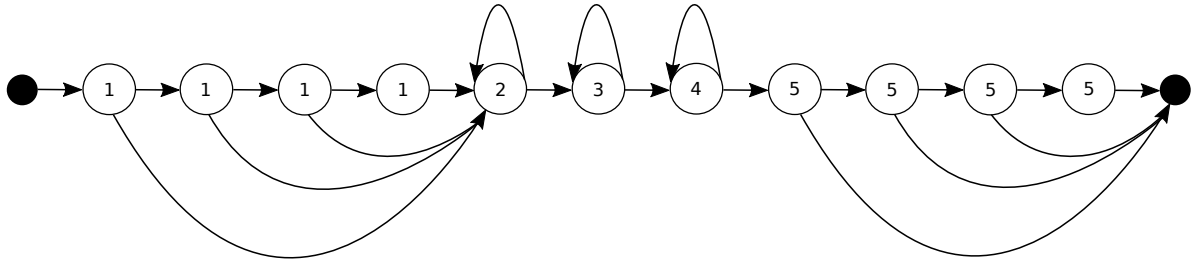


Obrázek 3.4: HMM: a) dlouhé pauzy (SIL) , b) krátké pauzy (SP).

trvání dané řečové jednotky a je také popsána například v [16], kde byla použita topologie s rozšířením pouze o jeden krajní stav. V obou pracích ([20] a [16]) bylo dosaženo celkové zlepšení přesnosti segmentace, při použití modifikovaných topologií.

3.4 Trénování HMM

V této části si popíšeme postupy, jaké se používají při trénování HMM. „Trénováním“, myslíme odhadování parametrů jednotlivých modelů a to přímo z řečových dat. Jelikož se jedná o statistickou metodu, tak její výsledky, mimo jiné, závisí na velikosti trénovacích dat, neboli v případě segmentace řeči na množství trénovacích promluv. Platí, že čím více máme těchto dat k dispozici, tím lepší odhady parametrů modelů můžeme získat. O jaké



Obrázek 3.5: Modifikovaný 5-ti stavový levo-pravý Markovův model se třemi emitujícími stavy a rozšířením krajních stavů (převzato z [20]).

parametry se jedná bylo popsáno v části 3.2 (vztah 3.7). Existují dva základní přístupy k trénování HMM. Prvním z nich je trénování *izolovaných jednotek*, při kterém jsou parametry HMM trénovány pomocí izolovaných segmentů řečového signálu. V tomto případě je však nutné znát přesné hranice segmentů v řeči, nebo je nutné mít tyto segmenty nahrány odděleně, což většinou není předem k dispozici. Druhou možností je trénování HMM za pomoci *vložených jednotek*, kdy se nejprve vytvoří odpovídající složený model promluvy z jednotlivých modelů a parametry těchto modelů jsou trénovány v rámci trénování modelu celé promluvy. Základním problémem při trénování HMM je inicializace modelů, kterou lze provést několika způsoby.

3.4.1 Inicializace HMM

Čím přesněji jsme schopni inicializovat HMM, tím přesnější odhad parametrů jednotlivých HMM dostaneme, jak je ukázáno v [12]. Jedny z nejpoužívanějších technik inicializace modelů jsou následující:

Flat-start, neboli „plochý start“ je nejjednodušší metoda pro inicializaci HMM. Používá se v případě, že nemáme k dispozici žádné předsegmentované promluvy, ani žádný soubor již natrénovaných modelů. Při flat-startu se parametry všech HMM nastavují na stejné hodnoty, například podle globální střední hodnoty a variance.

Bootstrap – ruční segmentace je metoda, kterou využíváme v případě, že máme k dispozici soubor předsegmentovaných promluv, tedy soubor ve kterém byly ručně vymezeny hranice jednotlivých řečových jednotek. Jak velký by měl tento soubor být záleží na konkrétní úloze, obecně lze říci, že bývá zpravidla mnohem menší (v řádu několika desítek promluv), než soubor pro trénování a to hlavně díky složitosti provádění ruční segmentace. Ovšem jak můžeme vidět v [12], i relativně malý soubor předsegmentovaných dat přináší přesnější inicializaci HMM a tím i přesnější odhad jejich parametrů, než při použití plochého startu.

Bootstrap - SI HMM - tato metoda je alternativou k metodě bootstrapu s využitím ručních segmentací. Využívá se zde postupů z oblasti automatického rozpoznávání řeči, kdy se určí nejpravděpodobnější hranice řečových jednotek na základě známé fonetické transkripce a vektoru parametrů dané promluvy, což pak lze použít pro inicializaci HMM. Více se o této metodě lze dozvědět v [19] a [12].

3.4.2 Trénování monofonových modelů

Po fázi inicializace HMM, přichází první fáze jejich trénování, tedy odhadování parametrů jednotlivých modelů. Nejčastější řečovou jednotkou, která se dnes používá při syntéze řeči a kterou tedy budeme používat i při automatické segmentaci je trifon (více v části 2.4.2). Postup při trénování HMM parametrů je takový, že se nejprve natrénují kontextově nezávislé řečové jednotky – monofony a na základě jejich „hrubé“ segmentace jsou následně inicializovány a trénovány kontextově závislé řečové jednotky – trifony.

Pokud bychom měli HMM pouze s jedním stavem, byl by odhad parametrů modelu velmi jednoduchý. Při zavedení vektoru pozorování ve tvaru $O_j = o_{j1}, o_{j2}, \ldots, o_{jT}$ u kterého předpokládáme, že byl generován stavem j, by se odhad střední hodnoty výstupní hustotní funkce tohoto stavu vypočítal podle vzorce:

$$\hat{\mu_j} = \frac{1}{T}\sum_{t=1}^{T} o_{jt}, \tag{3.8}$$

a odhad kovarianční matice výstupní hustotní funkce tohoto stavu by se vypočítal podle vzorce:

$$\hat{C_j} = \frac{1}{T}\sum_{t=1}^{T}(o_{jt} - \mu_j)(o_{jt} - \mu_j)^T. \tag{3.9}$$

HMM však většinou obsahuje více, než jeden stav, což bylo ukázáno v části 3.3. Navíc se jako výstupní hustotní funkce často používají hustotní směsi a je proto nutné použít nějaké jiné způsoby výpočtu parametrů modelu. Jelikož v průběhu tréninku dochází k postupnému zpřesňování parametrů modelu, jedná se o iterační cyklus a je tedy možné, zde použít iterační procedury. Při trénování HMM se nejčastěji využívá *metoda maximální věrohodnosti* a pro maximalizaci věrohodnostní funkce je výhodné využít *Baumův-Welchův reestimační algoritmus*.

Baumův-Welchův algoritmus se snaží v každém kroku najít lepší odhad parametrů, než které získal v kroku předchozím. Celý proces tedy končí ve chvíli, kdy jsou nové parametry stejné jako parametry předchozí. Tento algoritmus je obecně schopen nalézt pouze lokální minimum a záleží tak na počáteční inicializaci HMM, což už bylo nastíněno v části 3.4.1. Parametry výstupních hustotních funkcí jednotlivých stavů modelů je pak možné určit podle následujících vzorců:

Střední hodnoty

$$\bar{\mu}_{jm} = \frac{\sum_{t=1}^{T} L_j(t) o_t}{\sum_{t=1}^{T} L_j(t)}, \tag{3.10}$$

Váhy jednotlivých složek hustotních směsí

$$\bar{c}_{jm} = \frac{\sum_{t=1}^{T} L_j(t)}{\sum_{t=1}^{T} L_j(t)}, \tag{3.11}$$

Kovarianční matice

$$\bar{C}_{jm} = \frac{\sum_{t=1}^{T} L_j(t)(o_t - \bar{\mu}_{jm})(o_t - \bar{\mu}_{jm})^T}{\sum_{t=1}^{T} L_j(t)}. \tag{3.12}$$

$L_j(t)$ označuje pravděpodobnost, že jsme ve stavu j v čase t. Tuto pravděpodobnost je možné vypočítat s pomocí algoritmu *forward-backward*, jak je ukázáno například v [23]. Vztahy pro výpočet pravděpodobnosti přechodů mezi jednotlivými stavy modelů HMM, zde z důvodu jejich větší složitosti neuvádíme, ale lze je snadno dohledat například v [19], stejně jako postupné odvození výše uvedených vzorců.

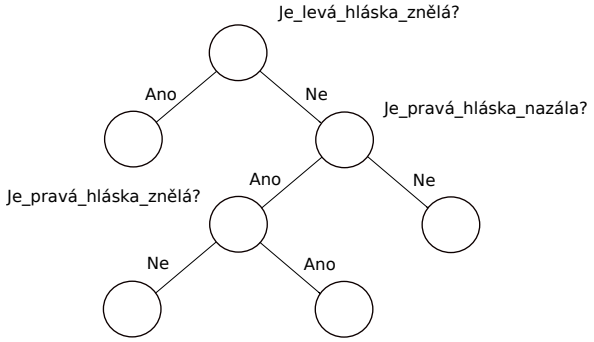
3.4.3 Trénování trifonových modelů

Jak už bylo zmíněno, monofonové jednotky jsou kontextově nezávislé. To přináší výhodu v jejich vcelku malém množství, ovšem také značnou nevýhodu v podobě zanedbání okolního kontextu. Výslovnost menší řečové jednotky jako hlásky, či fonému totiž silně závisí na okolí, v jakém se v textu nachází. Jistě si tedy všimneme, že kontextově nezávislé řečové jednotky nemohou dostatečně dobře postihnout jevy spojené s okolním kontextem, jak už bylo zmíněno v části 2.4. Tato nevýhoda je vyřešena zavedením jiné, kontextově závislé řečové jednotky *trifonu*. Jak už bylo řečeno v části 2.4.2, v každém jazyce existuje obrovské množství trifonů a natrénovat je všechny je v podstatě nemožné. Proto se trifony shlukují a vytvářejí se tak *zobecněné trifony*, nebo častěji *trifony se sloučenými stavy*.

Způsob trénování shluknutých trifonových modelů je pak v podstatě stejný jako v případě monofonových modelů.

Důležitou částí při vytváření trifonových modelů je tedy metoda shlukování. Těchto metod existuje samozřejmě více, ale zde je popsána jedna metoda, která je často využívána a jedná se o *shlukování založené na fonetických rozhodovacích stromech* (viz obrázek 3.6).

Rozhodovací stromy se v případě této metody konstruují pro všechny jednotlivé stavy modelů. Na začátku shlukování jsou nalezeny všechny trifony, které lze ze zvoleného monofonu odvodit na základě okolního kontextu. Například pro monofon „a“ to mohou být trifony *b-a+k, p-a+k*. Soubor všech takových trifonů je poté rozdělen do dvou subsouborů pomocí nějaké hodnoticí funkce a vhodně zvolených otázek. Tyto otázky jsou nejčastěji jednoduché, pouze s dotazem na okolní kontext a odpovědí na ně je buď ano, nebo ne, jak můžeme vidět na obrázku 3.6. Při každém dělení se sleduje rozdíl hodnoticí funkce před a po dělení u každé položené otázky. Otázka, která poskytne největší příspěvek k dělení souboru je následně k tomuto rozdělení použita. Je tedy žádoucí nejdříve pokládat takové otázky, které soubor rozdělí co možná nejvíce, tedy otázky s největším rozdílem hodnoticí funkce před a po dělení. Po dokončení rozhodovacího stromu vzniknou listy, u kterých už položení jakékoli otázky nevede k žádoucímu větvení, což je zpravidla určeno tak, že přírůstek hodnoticí funkce klesne pod předem stanovený práh. Listy rozhodovacích stromů představují stavy foneticky podobných trifonů, které mohou být při následném trénování shluknuty. Více o fonetických rozhodovacích stromech se lze dozvědět například v [19] a [23].



Obrázek 3.6: Jednoduchý rozhodovací strom pro shlukování trifonových modelů.

3.5 Segmentace pomocí HMM

Po natrénování modelů přichází na řadu samotná fáze segmentace. Ta spočívá v přiřazení jednotlivých HMM určitým sekvencím vektorů parametrů, čímž se v řečových promluvách identifikují úseky těmito HMM modelované a je tedy možné mezi nimi nastavit hranice. Jelikož počet sekvencí stavů je obrovský, využívá se zde techniky dynamického programování a k nalezení nejlepší sekvence stavů se používá *Viterbiův algoritmus*.

Viterbiův algoritmus je tedy schopný najít pravděpodobnost ($\varphi_j(t)$) maximálně pravděpodobné posloupnosti stavů, při použití daného modelu (λ) a při daném vektoru pozorování ($o_1 \dots o_i$) v časovém kroku t:

$$\varphi_j(t) = \max_{s(1),\dots,s(t-1)} P(o_1 \dots o_i, s(1), s(2), \dots, s(t) = s_j|\lambda). \tag{3.13}$$

Viterbiův algoritmus pracuje rekurzivně. Po počáteční inicializaci se v každém dalším časovém kroku ($t = 2, 3, \dots, T$) pro každý stav ($j = 2, \dots, N-1$) najde maximální hodnota pravděpodobnosti z předchozího kroku podle:

$$\varphi_j(t) = \{\max_{i=2,\dots,N-1} [\varphi_i(t-1)a_{ij}]\} b_j(o_t), \tag{3.14}$$

dále je zavedena speciální proměnná $\psi_j(t)$:

$$\psi_j(t) = \operatorname*{argmax}_{i=2,\dots,N-1} [\varphi_i(t-1)a_{ij}]\,. \tag{3.15}$$

Tato proměnná poté slouží k určení optimální posloupnosti stavů jejím zpětným trasováním. Více se lze dozvědět například v [19].

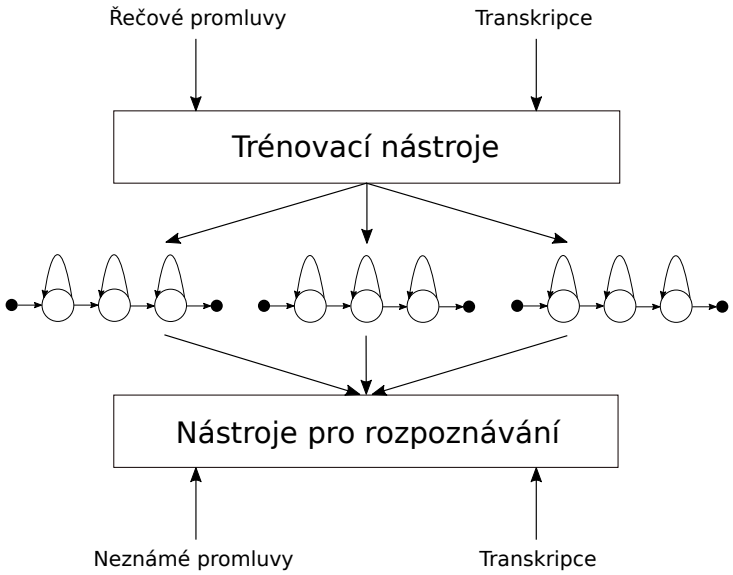
Kapitola 4

Nástroje pro automatickou segmentaci

V této části si popíšeme některé nástroje, s jejichž pomocí lze provádět automatickou segmentaci. Vzhledem k povaze této práce jsou zde popisovány hlavně dva nástroje a těmi jsou Hidden Markov Model Toolkit (dále jen HTK) a Kaldi. V části 4.3 je uvedeno několik dalších nástrojů, které mohou být také použity pro účely automatické segmentace, ale aktivně se jim v této práci nevěnujeme.

4.1 HTK

HTK je nástroj, primárně určený pro rozpoznávání řeči, který však nachází využití i v některých podpůrných úlohách syntézy řeči. Využívá statistické modelování řeči pomocí HMM. HTK může být nejjednodušeji rozděleno na části, které slouží k tréninku, neboli odhadování parametrů HMM a na části sloužící k rozpoznávání neznámých promluv. Toto rozdělení můžeme názorně vidět na obrázku 4.1. Funkce HTK jsou vystavěné na vzájemně spolupracujících knihovnách a nástrojích napsaných v jazyce C. Každá z těchto knihoven v sobě uchovává soubor dalších mnoha funkcí. K jakému účelu každá knihovna slouží, se lze dozvědět z rozsáhlé dokumentace v [25]. HTK nástroje jsou navrženy pro standardní spuštění přes příkazovou řádku a je možné jim zadat různé množství dodatečných argumentů, které mohou být obecné, nebo specifické.



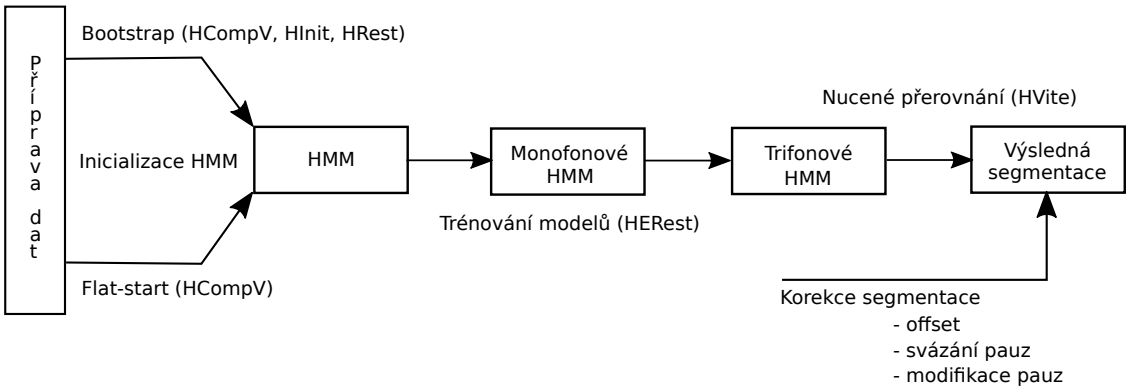
Obrázek 4.1: Základní schéma HTK (převzato z [25]).

4.1.1 Automatická segmentace

Na obrázku 4.2 můžeme vidět zjednodušené schéma automatické segmentace při použití nástroje HTK. Tento postup byl použitý při původním návrhu automatické segmentace v HTK, ze kterého tato práce vychází a nyní si popíšeme jeho jednotlivé části.

- **Příprava dat** – zahrnuje například vytvoření MFCC parametrů řečových promluv, fonetické transkripce, slovníku, topologie HMM, nebo konfiguračních souborů.
- **Inicializace HMM** – slouží k počátečnímu odhadu parametrů HMM. Jednotlivým možnostem inicializace HMM jsme se věnovali v části 3.4.1. V základním nastavení jsou HMM incializovány metodou bootstrapu, tedy za pomoci ruční segmentace. K tomu je využita trojice HTK nástrojů HCompV, HInit, a HRest. HCompV vytvoří počáteční modely na základě zadané topologie. Dvojice nástrojů HInit a HRest následně podle ručně nasegmentovaných promluv zpřesní počáteční parametry HMM před samotným procesem trénování. Další možností inicializace je metoda flat-start, při které je použit pouze nástroj HCompV, který nastaví parametry všech modelů na stejné hodnoty.
- **Trénování HMM** – je rozděleno na dvě fáze, jak bylo popsáno v části 3.4. Inicializované modely jsou použity pro trénování monofonových HMM nástrojem HERest. Na základě monofonových modelů jsou vytvořené trifonové modely, které jsou shlukovány pomocí fonetických rozhodovacích stromů. Takto shluknuté trifonové modely jsou následně trénovány opět nástrojem HERest.

- **Výsledná segmentace** – je získána pomocí metody „nuceného přerovnání“ (z anglického spojení *forced alignment*) známé z rozpoznávání řeči. Při této metodě je použit Viterbiův algoritmus, v podobě nástroje HVite, který má díky fonetické transkripci informaci o tom, jaká slova, respektive hlásky se nachází v řečových promluvách. Algoritmus poté neslouží k rozpoznání obsahu řečové promluvy, ale k přerovnání daných modelů tak, aby nalezl časové hranice zvolených řečových jednotek. Tato segmentace je následně dodatečně upravována tak, aby bylo dosaženo ještě vyšší přesnosti hranic. Tyto úpravy jsou popsány v části 5.6.



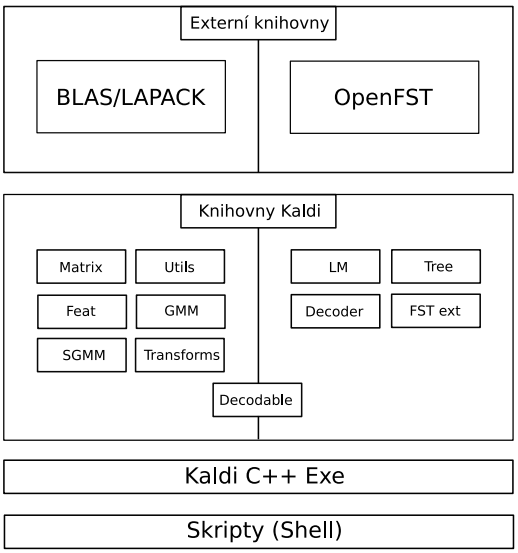
Obrázek 4.2: Automatická segmentace v HTK.

4.2 Kaldi

Kaldi je open-source nástroj napsaný v C++ primárně určený pro aplikace spojené s rozpoznáváním řeči, založený na vážených konečných transducerech (z anglického spojení Weighted Finite State Transducers, dále jen WFST). S WFST pracuje skrze svoje vlastní knihovny, nebo s použitím obecně známého nástroje OpenFst. Kaldi je licencováno pod Apache License v2.0 a jeho obsah je volně dostupný. Standardně se použití Kaldi doporučuje především na unixových systémech.

Základní popis a cíle Kaldi se dají srhnout do několika bodů :

- Využití WFST – Kaldi WFST skripty a OpenFst
- Podpora knihoven lineární algebry – BLAS a LAPACK
- Open–source licence
- Kompletní recepty pro sestavení systémů zaměřených především na oblast zpracování řeči



Obrázek 4.3: Základní pohled na nástroj Kaldi (převzato z [18]).

Jak můžeme vidět na obrázku 4.3, jednotlivé moduly Kaldi mohou být rozděleny do dvou skupin. První skupina představuje moduly, které využívají knihovny lineární algebry (BLAS, LAPACK) a druhá skupina představuje moduly, které využívají knihovny OpenFst. Pro komunikaci mezi těmito dvěma částmi je možné využít modul *Decodable*. K jednotlivým funkcím se přistupuje skrze nástroje příkazové řádky napsané v C++, které jsou volány ze skriptů. Každý nástroj má specifickou funkci, což přispívá k vysoké modularitě a tedy i k snadné implementaci nových algoritmů. Kaldi je, na rozdíl od HTK, v dnešní době stále intenzivně vyvíjeno, čímž se stává dobrým kandidátem na vytváření nových řečových systémů. Bohužel, v současné době Kaldi postrádá kompletní dokumentaci, a tak se lze pro více základních informací odkázat především na [18].

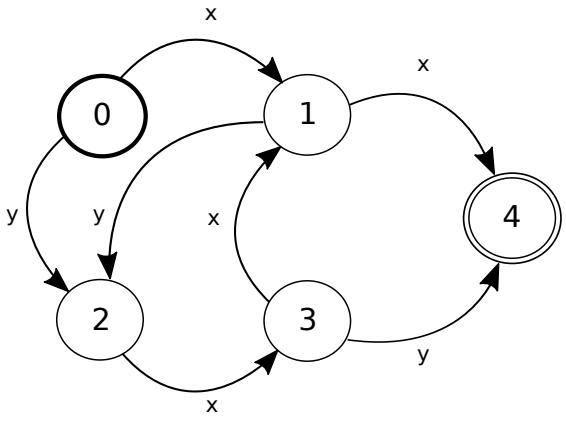
Jak už jsme se zmínili, Kaldi velmi často využívá WFST (více v části 4.2.1), ať už se jedná o trénování HMM, segmentaci řečových jednotek, nebo v případě rozpoznávání řeči o dekódování. Jako WFST je podle [15] také možné modelovat například HMM, slovníky, nebo gramatiky (při rozpoznávání řeči). Základním přístupem při práci s Kaldi je tedy převedení uživatelských dat (řečových promluv, transkripce, atd.) do formátu, ze kterého je možné vytvořit WFST. Celý proces je detailněji popsán v části 5 při konkrétní implementaci naší úlohy.

4.2.1 Vážené konečné transducery (WFST)

Vážené konečné transducery jsou speciální typy konečných automatů, od kterých se liší několika způsoby, které si nyní představíme.

Na obrázku 4.4 je uveden příklad konečného automatu popsaného stavovým diagramem. Konečný automat lze použít pro zjištění, zda vstupní řetězec, který na vstup to-

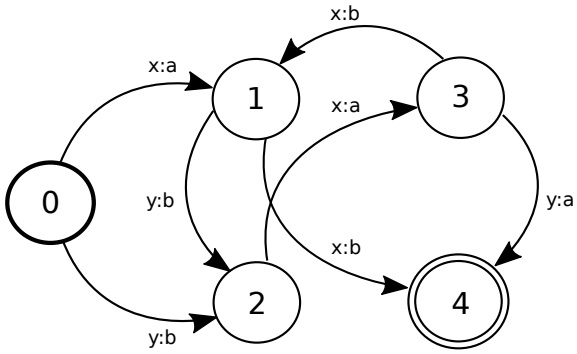
hoto automatu přivedeme, patří do daného jazyka[1]. Vstupní řetězec je postupně zpracováván konečným automatem. Každý znak vstupního řetězce převede konečný automat do nějakého stavu, podle symbolů uvedených u jednotlivých přechodů. Pokud se objeví ve vstupním řetězci znak, který automat nezná, nebo pro něj není z aktuálního stavu definován přechod do dalšího stavu, je vstupní řetězec automaticky nepřijímán. Pokud vstupní řetězec převede automat z počátečního stavu do jednoho z jeho koncových stavů, pak je tímto automatem přijímán. Například konečný automat z obrázku 4.4 bude přijímat řetězce $xx, xyxy, yxxx$, ale nepřijme řetězce yy, nebo xyy. O konečných automatech se lze více dozvědět například v [4].



Obrázek 4.4: Příklad konečného automatu (tučně vyznačený kruh = počáteční stav, dvojitý kruh = konečný stav).

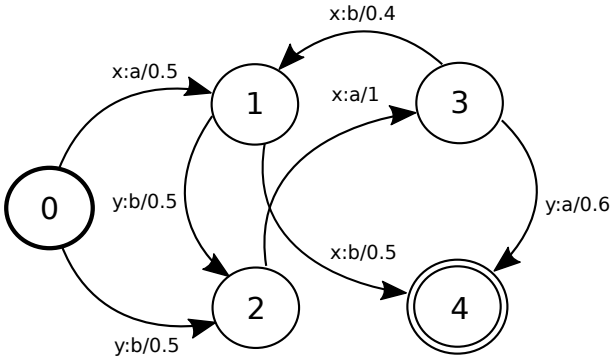
Pokud konečný automat obohatíme o výstupní symboly u jednotlivých přechodů, dostaneme *konečný transducer* (z anglického spojení Finite-State Transducer – FST). V takovém případě cesta skrze stavy automatu přemapuje symboly vstupního řetězce na sekvenci výstupních symbolů. Pokud bychom takto upravili automat z obrázku 4.4, výsledkem by mohl být například konečný transducer na obrázku 4.5.

[1]V případě použití konečného automatu máme na mysli regulární jazyky.



Obrázek 4.5: Příklad konečného transduceru (tučně vyznačený kruh = počáteční stav, dvojitý kruh = konečný stav).

Pokud konečný transducer doplníme o pravděpodobnosti jednotlivých přechodů, dostaneme *vážený konečný transducer* (WFST), který můžeme vidět na obrázku 4.6. O WFST se lze více informací dozvědět například v [15].



Obrázek 4.6: Příklad váženého konečného transducera (tučně vyznačený kruh = počáteční stav, dvojitý kruh = konečný stav)

4.3 Další nástroje

4.3.1 CMU Sphinx

CMU Sphinx je open-source program vyvinutý univerzitou Carnegie Mellon, laboratoří Sun Microsystems a laboratoří Mitsubishi Electric Research, určený pro rozpoznávání řeči. Je napsaný v jazyce Java a zaměřuje se především na praktické úlohy. Využívá statistický

přístup k modelování řeči pomocí HMM. V [8] je uveden nástroj pro automatickou segmentaci s použitím části CMU Sphinx nazvané SphinxTool.

4.3.2 Julius

Julius je vysoce výkonný open-source software pro rozpoznávání řeči určený především pro výzkum. Běží pod unixovými systémy a částečně i na Windows. Podporuje standardní přístup modelování řeči pomocí HMM s různými počty směsí v hustotních funkcích. Původní modely pod tímto nástrojem byly vyvinuty pro japonštinu, ale postupně se adaptovaly i na ostatní jazyky. V [10] je ukázáno, že Julius dosahuje dobré úspěšnosti především v rozpoznávání řeči.

SPPAS – nástroj pro vytvoření a vizualizaci anotací řečových promluv. Je napsaný v jazyce Python a lze jej spustit ve Windows, unixových systémech i na Mac OS X. SPPAS je schopný provádět automatickou segmentaci řeči na úrovni vět, slabik a hlásek na základě nahraných promluv a jejich transkripcí. Jeho algoritmy pro automatickou segmentaci řeči jsou založené především na postupech, které používá Julius a některých funkcích z HTK. Má uživatelsky přívětivé grafické prostředí a je určen především jako pomůcka pro lingvisty. Více se lze o tomto nástroji dozvědět v [1].

4.3.3 Praat

Praat je volně dostupný fonetický nástroj pro analýzu a syntézu řeči vyvíjený od roku 1992 na univerzitě v Amsterdamu. Podporuje grafické prostředí, které je schopné vizualizovat řečové promluvy. Syntézu provádí pomocí techniky zdroje a filtru, generováním různých signálů, nebo pomocí artikulační syntézy, kde výslednou řeč generuje na základě fyziologických informací. Nepodporuje techniku převodu textu na řeč a je využívaný především fonetiky pro svou názornost a jednoduchost. Více se lze dozvědět například v [2].

EasyAling – volně dostupný program vytvořený jako plugin do nástroje Praat. Je schopný provádět automatickou fonetickou segmentaci řečových promluv na základě jejich transkripce v textovém formátu. Výslednou segmentaci je možné dodatečně upravovat v grafickém uživatelském prostředí pro dosažení vyšší kvality časových hranic. Více informací o tomto nástroji se lze dozvědět například v [5].

Labtool – je nástroj pro segmentaci založený na HMM. Tento nástroj je, mimo jiné, navržen pro použití v nástroji Praat . Více informací o nástroji Labtool lze nalézt například v [17], kde je i popsána možnost převedení stávajících funkcí (z HTK) tohoto nástroje do Kaldi.

Kapitola 5

Praktická realizace v Kaldi

V této části uvedeme podrobný postup, který byl použit při praktické realizaci automatické fonetické segmentace řeči za pomoci nástroje Kaldi. Tento návrh byl inspirován volně dostupnými příklady[1] z Kaldi, které lze nalézt v adresáři Kaldi.

Jak už jsme se zmínili v úvodu, jedním z úkolů této práce je převést algoritmy automatické segmentace vytvořené v HTK (popsané v části 4.1.1) do Kaldi. Kaldi ovšem pracuje odlišně, než HTK a proto není možné převést algoritmy přesně a je nutné v Kaldi celý proces automatické segmentace znovu vytvořit. Nejprve se zde budeme věnovat přípravě vstupních souborů (části 5.2.1 a 5.2.2). Dále se zaměříme na vytvoření topologie HMM (část 5.3) a trénování těchto modelů s jejich následnou segmentaci (část 5.5). V závěru se zaměříme na modifikace tohoto návrhu použité pro zvýšení přesnosti celkové automatické segmentace (část 5.7).

5.1 Použitý řečový korpus

Řečový korpus, který byl v této práci použit má následující parametry:

- Mužský hlas (1 řečník)
- 12 424 řečových promluv (přibližně 18,5 hodin řeči)
- Promluvy nahrány ve stylu „čtení zpráv“
- 12 424 vět
- celková délka korpusu (včetně pauz): 59797 s
- celková délka korpusu (bez pauz): 53055 s

O tomto korpusu se lze více informací dozvědět například z [14], kde je i popsán způsob jeho nahrávání.

[1] Z příkladů byl využit především systém pro rozpoznávání řeči pracující s korpusem řečových promluv z Wall Street Journal (WSJ).

5.2 Příprava vstupních dat

V této části se budeme věnovat přípravě vstupních dat, což zahrnuje vytvoření několika souborů ve formátu čitelném pro Kaldi. Tato data jsou v základním nastavení použity ve formě vektorů MFCC parametrů (12 statických parametrů + delta + delta–delta koeficienty = 39 koeficientů).

5.2.1 Příprava datových souborů

Jak už bylo zmíněno, Kaldi vyžaduje jistou specifikaci vstupních dat, které chceme použít. V této části si popíšeme především specifikaci vektorů MFCC parametrů řečových promluv a jejich transkripcí, což zahrnuje vytvoření určitých souborů, které si nyní uvedeme. Celá datová složka se všemi vytvořenými položkami (standardně označovaná jako `/data`) může obsahovat až 8 souborů:

- cmvn.scp
- feats.scp
- reco2file_and_channel
- segments
- spk2utt
- text
- utt2spk
- wav.scp

Z těchto souborů, je ovšem třeba „ručně“ vytvořit pouze některé z nich. Pro tento účel byl v této práci vytvořen skript, který lze nalézt ve složce `local` pojmenovaný `prepare_data.sh`. Ten obstarává vytvoření všech potřebných souborů, které si v této části uvedeme. Ostatní soubory se standardně vytvářejí pomocí skriptů, které jsou součástí Kaldi.

utt2spk (neboli utterance to speaker) je textový soubor, který má na každé řádce označení řečové promluvy ⟨*číslo–promluvy*⟩, společně s označením řečníka ⟨*číslo–řečníka*⟩. Pokud jsou ovšem promluvy namluvené pouze jedním řečníkem, je doporučováno použít strukturu:

⟨*číslo–promluvy*⟩ ⟨*číslo–promluvy*⟩.

Soubor *utt2spk* pro dvě promluvy s označením `doplň00003_00` a `doplň00005_00` tedy bude mít následující tvar:

dopln00003_00	dopln00003_00
dopln00005_00	dopln00005_00

text je textový soubor, který obsahuje přepis každé promluvy. Jeho struktura je:

⟨*číslo–promluvy*⟩ ⟨*přepis*⟩.

Na každé řádce se uvádí označení promluvy (⟨*číslo–promluvy*⟩) s textovým přepisem řečeného obsahu. Ten může být doplněn o dodatečné informace o neřečových událostech. V našem případě to byly především nádechy (_BREATH_). Jako příklad uvádíme soubor `text` pro řečové promluvy s označením `dopln00003_00` a `dopln00005_00`:

dopln00003_00 _BREATH_ Kolik kilogramů se tedy dá reálně zhubnout za měsíc?
dopln00005_00 _BREATH_ Z čeho tak usuzujete?

wav.scp[2] je textový soubor, který obsahuje cesty ke zvukovým souborům s jednotlivými promluvami, jeho struktura je:

⟨*číslo–promluvy*⟩ ⟨*cesta-k-souboru*⟩.

Jako příklad uvádíme soubor `wav.scp` pro dvě řečové promluvy s označením `dopln00003_00` a `dopln00005_00`:

dopln00003_00 wav_soubory/dopln00003_00.wav
dopln00005_00 wav_soubory/dopln00005_00.wav

segments[3] je textový soubor, který obsahuje hranice jednotlivých promluv v řečovém signálu. Používá se v případě, že se v jedné zvukové nahrávce nachází více promluv. Jeho struktura je:

⟨*číslo–promluvy*⟩ ⟨*číslo–nahrávky*⟩ ⟨*počátek–segmentu*⟩ ⟨*konec–segmentu*⟩.

reco2file_and_channel[4] je textový soubor, který se používá, pokud se měří četnost chyb pomocí nástroje NIST – sclite.

Ostatní soubory je již možné vytvořit pomocí skriptů z Kaldi a není nutné je vytvářet ručně. Mezi ně patří:

cmvn.scp obsahuje cesty a klíče k archivu se statistikami pro kepstrální střední hodnoty a variance (z anglického spojení cepstral mean and variation (CMVN)). Jeho struktura je:

[2]Přípona .scp označuje skript, což je jeden ze souborů, který se používá při čtení, nebo zapisování dat v Kaldi.

[3]Soubor *segments* v této práci nebyl použit, protože máme každou řečovou promluvu uloženou v samostatném souboru.

[4]Soubor *reco2file_and_channel* v této práci nebyl použit, protože zde nepoužíváme nástroj NIST.

⟨*číslo–promluvy*⟩ ⟨*cesta–k–souboru*⟩.

Tento soubor je automaticky vytvořen skriptem `compute_cmvn_stats.sh`, který CMVN počítá. Jako příklad uvádíme soubor `cmvn.scp` pro dvě řečové promluvy s označením `dopln00003_00` (uložena v archivu pod klíčem 14) a `dopln00005_00` (uložena v archivu pod klíčem 267).

dopln00003_00 data/mfcc/cmvn_train.ark:14
dopln00005_00 data/mfcc/cmvn_train.ark:267

feats.scp obsahuje cesty a klíče k archivu s uloženými parametry všech promluv. Jeho formát je:

⟨*číslo–promluvy*⟩ ⟨*cesta–k–souboru*⟩.

Tento soubor je automaticky vytvářen skriptem `make_mfcc.sh` v případě MFCC parametrizací, nebo skriptem `make_plp.sh` v případě PLP parametrizací. Jako příklad uvádíme soubor *feats.scp* pro dvě řečové promluvy s označením `dopln00003_00` (uložena v archivu pod klíčem 14) a `dopln00005_00` (uložena v archivu pod klíčem 32023) při použití vektorů MFCC parametrů.

dopln00003_00 data/mfcc/raw_mfcc_train.1.ark:14
dopln00005_00 data/mfcc/raw_mfcc_train.1.ark:32023

5.2.2 Příprava jazykových souborů

Přípravou jazykových souborů rozumíme vytvoření několika souborů s informacemi o použitých hláskách, slovníku, pauzách, neřečových událostech, topologie HMM, atd. Nejedná se tedy o vytváření gramatik, které se standardně používají při rozpoznávání řeči. Složka se všemi potřebnými jazykovými soubory (standardně označovaná jako `/lang`) obsahuje 10 položek, kterými jsou:

- dict
- phones
- tmp
- L.fst
- L.disambig.fst
- oov.int
- oov.txt
- phones.txt

- topo
- words.txt

Stejně jako v případě přípravy vstupních dat, i zde není potřeba všechny soubory vytvářet „ručně“. Pro tento účel byl v této práci vytvořen skript `prepare_lang.sh` (ze složky `local`), který vytváří všechny potřebné soubory. Všechny soubory vytvořené tímto skriptem jsou ukládány do složky `/lang/dict`, která musí obsahovat následující položky:

- lexicon.txt
- extra_questions.txt
- nonsilence_phones.txt
- optional_silence.txt
- silence_phones.txt

Nyní si upřesníme význam jednotlivých souborů a uvedeme si jejich strukturu.

lexicon.txt je slovník obsahující fonetické výslovnosti jednotlivých slov v podobě sekvencí hlásek. Pro každé slovo se zde může nacházet více, než jedna výslovnost. Jeho struktura je:

$$\langle \text{slovo} \rangle \ \langle \textit{hláska1} \rangle \ \langle \textit{hláska2} \rangle \ \ldots \ \langle \textit{N–tá hláska} \rangle.$$

V základním nastavení naší automatické segmentace je každé slovo ve slovníku doplněno o možnost přidané pauzy (SIL) na konci tohoto slova. Jak tato pauza vypadá je podrobněji popsáno v části 5.3. Jako příklad, zde uvedeme soubor `lexicon.txt` pro slova *automatu* a *automechanik*, kde můžeme vidět verze slov bez pauzy i s pauzou.

automatu	Y t o m a t u
automatu	Y t o m a t u SIL
automechanik	Y t o m e x a n i k
automechanik	Y t o m e x a n i k SIL

extra_questions.txt obsahuje dodatečné otázky pro vytváření rozhodovacích fonetických stromů při shlukování trifonů. Vytvářením fonetických rozhodovacích stromů se důkladněji věnujeme v části 5.5.2. Vzhledem k tomu, že Kaldi si otázky vytváří automaticky, je možné tento soubor ponechat prázdný. Otázky jsou zde reprezentovány posloupností čísel, kde každé číslo označuje danou hlásku (toto mapování je provedeno pomocí souboru `phones.txt`, který je definován později). Struktura každé řádky souboru `extra_questions.txt` je následující:

$$\langle \textit{číslo–první–hlásky} \rangle \ \langle \textit{číslo–druhé–hlásky} \rangle \ \ldots \ \langle \textit{číslo–N–té hlásky} \rangle.$$

Tento soubor v základním nastavení automatické segmentace nepoužíváme. Je však důkladněji popisován v části 5.7, kde se diskutuje přínos používání předvytvořených otázek, místo automaticky generovaných otázek pomocí Kaldi.

nonsilence_phones.txt obsahuje všechny znaky, které nepředstavují pauzy a neřečové události. Většinou se sem tedy umisťují všechny použité hlásky, které můžeme označit jako „skutečné“ hlásky, neboli písmena abecedy. Na každé řádce se nachází jedno písmeno.

optional_silence.txt obsahuje jedinou řádku s pauzou, označenou SIL a to i v případě, že používáme více, než jednu pauzu.

silence_phones.txt obsahuje všechny použité pauzy, včetně pauzy SIL. Jak je uvedeno v části 5.3, v základním nastavení zde používáme pouze jednu pauzu a tou je SIL. Dále se zde nachází znak SPN, což je v Kaldi standardní označení pro krátký šum. Ten se používá jako realizace tzv. mimoslovníkového výrazu, který je zde definován později. Poslední položkou v souboru `silence_phones.txt` je znak %, který zde byl použit pro hláskovou reprezentaci nádechu (v přepisu řečových promluv označovaném jako _BREATH_).

Ostatní soubory vytvoříme za pomoci Kaldi skriptu `prepare_lang.sh` ze složky `utils`, který má tvar:

```
prepare_lang.sh lang/dict/ "<UNK>" lang/tmp/ lang/
```

Část `lang/dict` je vstupní adresář se soubory vytvořenými uživatelem, které jsme si definovali v předchozí části. Část `"<UNK>"`[5] je tzv. mimoslovníkový výraz označovaný jako OOV (z anglického spojení out-of-vocabulary). Slova, která se vyskytnou v transkripci, ale nemají své zastoupení ve slovníku, jsou namapována na tento výraz. Další část příkazu označená jako `lang/tmp` je pouze dočasná složka a dále se nepoužívá a poslední část `lang` je výstupní složka. Nyní si představíme soubory, které se v této složce vytvoří:

L.fst je slovník ve formátu FST,

phones.txt je seznam všech hlásek, včetně pauz a šumů v textové formě. Ke každé položce v tomto souboru je přiřazena číselná hodnota. Tento soubor poté slouží jako tabulka pro vytvoření FST v OpenFst formátu. Jako příklad si zde uvedeme část souboru `phones.txt`:

G	11
H	12
I	13
J	14

[5] V tomto tvaru se tento výraz zapisuje v Kaldi.

topo je topologie HMM v textové formě, kterou Kaldi ve standardním nastavení vytváří automaticky. Její podrobný popis je proveden v části 5.3.

words.txt je textový soubor, který obsahuje seznam všech slov. Každému slovu je zde přiřazena číselná hodnota. Tento soubor poté slouží jako tabulka pro vytvoření FST v OpenFst formátu. Jako příklad si zde uvedeme část souboru `words.txt`:

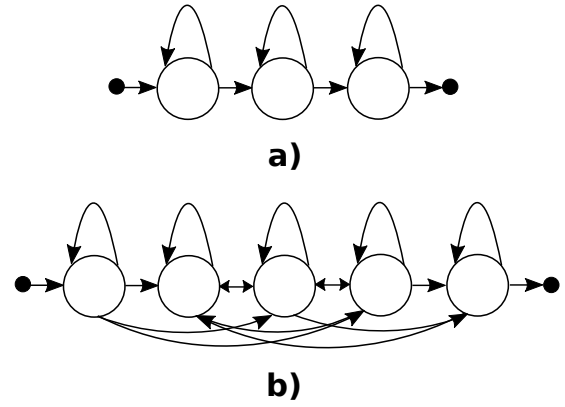
⋮	⋮
kadet	6698
kadeřnice	6699
kafe	6700
kafelnikov	6701
⋮	⋮

Skriptem `prepare_lang.sh` je dále vytvořena složka `/lang/phones`. Tato složka rovněž obsahuje velké množství souborů, které jsou většinou stejné jako ty, které jsme si už uvedli, pouze v odlišném formátu[6]. Dále se ve složce `/lang/phones` nachází několik souborů, kterými lze specifikovat způsob generování rozhodovacích stromů. Jelikož jsme v této práci jejich strukturu neměnili, jedná se tak většinou jen o další seznamy hlásek a jejich podobu si tu tedy neuvádíme.

5.3 Vytvoření topologie HMM

V základním nastavení používáme pro všechny hlásky klasický 5-ti stavový model (viz obrázek 5.1a) a pro pauzu je využit speciální model (viz obrázek 5.1b). Modely pauz se zde liší od modelů, které byly použity v původním návrhu v HTK. Návrh v HTK používal dva druhy pauzy a to krátkou, nebo také mezislovní (*SP*) a dlouhou (*SIL*), jejichž modely byly představeny v části 3.3. Model krátké pauzy, díky své topologii, dovoluje vložení krátké pauzy mezi každá dvě slova, ikdyž se mezi nima pauza ve skutečnosti nenachází. Pokud se zjistí, že mezi dvěma slovy pauza skutečně není, je jí nastavena nulová délka a model pauzy se, mezi těmito slovy, přeskočí. V Kaldi se ovšem nedá použít model krátké pauzy, protože Kaldi nedovoluje přechod z neemitujícího stavu do dalšího neemitujícího stavu. V Kaldi se proto používá speciální model pauzy *SIL*, který můžeme vidět na obrázku 5.1b. Tento model je nejčastěji používaným modelem pauzy při návrhu algoritmů v Kaldi. Při standardním nastavení je také automaticky generován skripty Kaldi a proto jsme ho pro základní experimenty použili i v naší práci.

[6]Každý ze souborů se může vytvořit až ve třech formátech : textový (.txt), seznam oddělovaný dvojtečkou (.csl) a číselný (.int).



a)

```
<Topology>
<TopologyEntry>
<ForPhones>
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.5 <Transition> 1 0.5 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.5 <Transition> 2 0.5 </State>
<State> 2 <PdfClass> 2 <Transition> 2 0.5 <Transition> 3 0.5 </State>
<State> 3 </State>
</TopologyEntry>
```

b)

```
<TopologyEntry>
<ForPhones>
1
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.25 <Transition> 1 0.25
<Transition> 2 0.25  <Transition> 3 0.25</State>
<State> 1 <PdfClass> 1 <Transition> 1 0.25 <Transition> 2 0.25
<Transition> 3 0.25  <Transition> 4 0.25</State>
<State> 2 <PdfClass> 2 <Transition> 1 0.25 <Transition> 2 0.25
<Transition> 3 0.25  <Transition> 4 0.25</State>
<State> 3 <PdfClass> 3 <Transition> 1 0.25 <Transition> 2 0.25
<Transition> 3 0.25  <Transition> 4 0.25</State>
<State> 4 <PdfClass> 4 <Transition> 4 0.5 <Transition> 5 0.5 </State>
<State> 5 </State>
</TopologyEntry>
</Topology>
```

Obrázek 5.1: Základní topologie použitých HMM modelů, společně s textovým souborem `topo` reprezentujícím topologii v Kaldi – a) klasický model použitý pro modelování hlásek, b) model použitý pro modelování pauzy.

Na obrázku 5.1 je rovněž ukázána textová forma HMM topologie, ve tvaru čitelném pro Kaldi. Pro definování HMM topologie se používá soubor *topo* ze složky `lang`, který jsme zmiňovali v části 5.2.2. Definice HMM topologie je podobná definici HMM topologie v HTK, od které se ovšem v několika bodech odlišuje. V následující tabulce jsou popsané její jednotlivé části.

$\langle Topology \rangle$	Úvodní značka, která označuje počátek celého souboru *topo*. Píše se pouze na počátku souboru a poté na jeho konci s ukončovací značkou $\langle /Topology \rangle$.
$\langle TopologyEntry \rangle$	Počáteční značka označující začátek topologie každého modelu. Na konci každé topologie se píše s ukončovací značkou $\langle /TopologyEntry \rangle$. V Kaldi se všechny topologie uchovávají v jednom textovém souboru.
$\langle ForPhones \rangle$	Za touto značkou následuje výpis všech hlásek, pro které je daná topologie modelu určena. Hlásky jsou zde reprezentovány svými číselnými formami (podle souboru `phones.txt`).
$\langle State \rangle$	Úvodní značka pro specifikaci stavu modelu dané topologie. Za touto značkou následuje číslo tohoto stavu, číslo *pdf-class* (specifikováno níže) a definice jednotlivých přechodů mezi stavy, označovaná jako *Transition* (specifikováno níže). Definice stavu je ukončena značkou $\langle /State \rangle$.
$\langle PdfClass \rangle$	Každý stav, každého modelu má proměnnou *pdf-class*, neboli číselné označení výstupní hustotní funkce tohoto stavu, která slouží k mnoha účelům. Standardně má každý stav modelu odlišnou hodnotu *pdf-class* (číslováno vzestupně od 0). Pokud mají dva stavy (nebo i více stavů) stejnou hodnotu u proměnné *pdf-class*, budou sdílet stejnou hustotní funkci a při trénování modelů se jejich parametry natrénují na stejné hodnoty. Proměnná *pdf-class* se dále používá pro označení emitujících a neemitujících stavů. Pokud je jako *pdf-class* zvoleno číslo -1, nebo je celá tato část vynechána, znamená to, že stav je neemitující.
$\langle Transition \rangle$	Používá se pro definici přechodů mezi jednotlivými stavy. Nejprve je uvedeno číslo stavu do kterého přechod směřuje a následně je definována pravděpodobnost tohoto přechodu.

5.4 MFCC parametrizace

Jako základní parametrizaci řečových souborů používáme MFCC parametry (více v části 2.7.1) s délkou okénka 25 ms a s posunem okénka 6 ms. Vygenerování MFCC parametrů v Kaldi obstarává skript `make_mfcc.sh`. Nachází se ve složce `steps` v adresáři nástroje Kaldi. Používá se v následujícím tvaru:

make_mfcc.sh `--mfcc-config conf/mfcc.conf data/train exp/make_mfcc/train featdir`

Prvním parametrem je konfigurační soubor `mfcc.conf` specifikující informace pro výpočet MFCC parametrů. Jeho struktura je:

htk-compat	Zaručí, že vypočtené parametry budou podobné parametrům získaným pomocí nástroje HTK. Ke standardním 12 statickým MFCC koeficientům se přidá vypočtená energie, takže získané MFCC koeficienty budou odpovídat koeficientům z HTK za použití nastavení MFCC_E_D_A .
use-energy=true	Použití energie místo C0 (nultý kepstrální parametr).
sample-frequency=16000	Vzorkovací frekvence vstupních řečových souborů.
dc–offset	Zaručí podobné nastavení, jako za použití příkazu ZMEANSOURCE v nástroji HTK.
frame–length=25	délka okénka
frame–shift=6	posun okénka
window–type=hamming	typ okénka

Dalším argumentem skriptu `make_mfcc.sh` je `data/train`, tedy složka s předvytvořenými datovými soubory, které byly popsány v části 5.2.1.

Výpočet MFCC parametrů ve skriptu `make_mfcc.sh` se provádí pomocí nástroje `compute-mfcc-feats`. Tento příkaz vyžaduje zadání vstupního souboru s daty (v Kaldi často označovaném jako *rspecifier*) a výstupního souboru (v Kaldi často označovaném jako *wspecifier*), kam budou zapsány vypočítané parametrizace. Jako *rspecifier* je použit soubor `wav.scp`, popsaný v časti 5.2.1. Vytvořené MFCC parametry jsou uložené do dvou souborů. Prvním z nich je binární archiv (.ark) a druhým je skript (.scp), který obsahuje adresu a klíče k parametrům uložených v archivu.

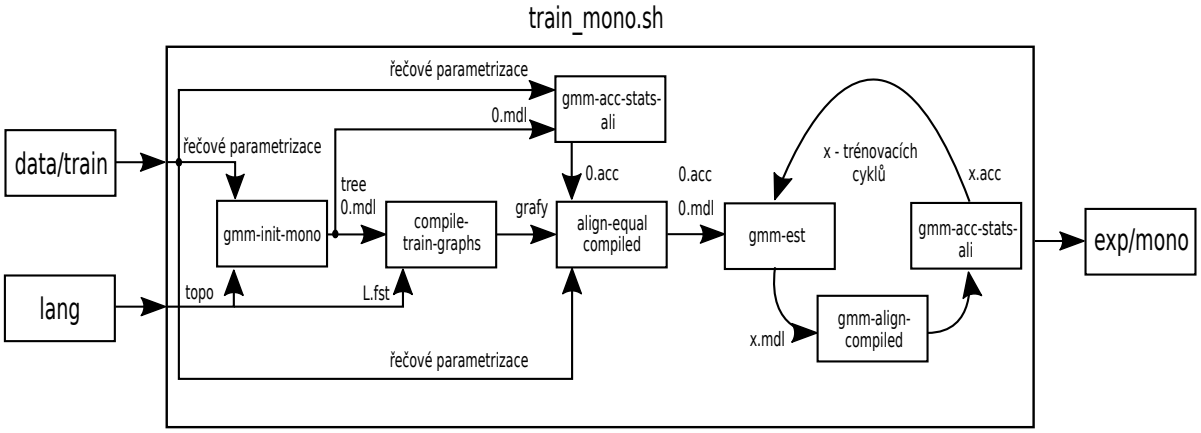
Další částí procesu parametrizace je vypočtení normalizovaných kepstrálních středních hodnot a variance (CMVN). Podle [22] je získání těchto hodnot vhodné pro potlačení vlivu šumu prostředí, kde byly náhravky pořízené, na výslednou parametrizaci. V našem případě byly nahrávky nahrané v nezašuměném prostředí (viz část 5.1) a proto by tyto hodnoty neměli mít na celkový proces vliv. Kaldi ovšem jejich vypočítání vyžaduje a proto byly do našeho algoritmu rovněž zahrnuty. Stejně jako v případě MFCC parametrů byly i CMVN uloženy do dvou souborů - archiv (.ark) a skript (.scp).

5.5 Trénování HMM

5.5.1 Trénování monofonových modelů

V této části je popsáno trénování monofonových, tedy kontextově nezávislých modelů. Před samotným trénováním je nutné modely nejprve inicializovat. Popis jednotlivých možností inicializace byl popsán v části 3.4.1. Bohužel v současné době není v Kaldi implementována

možnost využití bootstrapu, neboli ručních segmentací při inicializaci modelů a tak jsme použili nejjednodušší metodu *flat-startu*. Pro samotné trénování monofonových HMM se nevyužívá celý řečový korpus. Jak je ukázáno například v [21], k trénování monofonových HMM stačí přibližně 1000 promluv. Použití více promluv nevede ke zlepšení odhadu parametrů a proto i zde jich pro úsporu času více nepoužíváme. Skript, který slouží k trénování HMM se jmenuje `train_mono.sh`. Zjednodušené schéma vnitřní struktury tohoto skriptu můžeme vidět na obrázku 5.2:



Obrázek 5.2: Vnitřní struktura skriptu `train_mono.sh`.

Nástroj `train_mono.sh` je použit v následujícím tvaru:

train_mono.sh `data/train lang exp/mono`

První argument `data/train` je složka s datovými soubory, které byly popsány v části 5.2.1, `lang` je složka s jazykovými specifikacemi popsaná v části 5.2.2 a `exp/mono` je výstupní složka, která bude obsahovat natrénované monofonové modely.

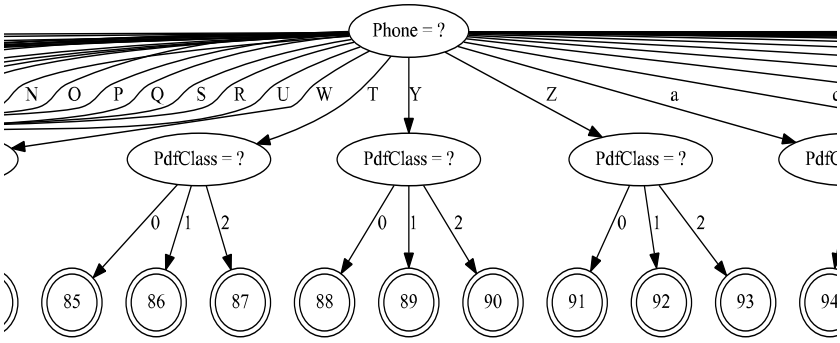
Jak můžeme vidět ze struktury na obrázku 5.2, skript `train_mono.sh` obsahuje několik dílčích nástrojů s odlišnými funkcemi. Ty nejdůležitější si nyní blíže popíšeme.

Prvním z těchto nástrojů je `gmm-init-mono`. Tento nástroj se používá pro počáteční inicializaci HMM modelů pomocí metody *flat-start*. V jistém smyslu tak zastává podobnou funkci jako nástroj HCompV v HTK. Používá se v následujícím tvaru:

gmm-init-mono `topo feat_dim 0.mdl tree`

Vstupními parametry jsou topologie HMM (`topo`) a dimenze vektoru parametrů řečového signálu (`feat_dim`). Výstupem je poté vytvořený model, zde označený jako `0.mdl` a rozhodovací fonetický strom `tree`. Když se podíváme na obrázek 5.3 vidíme, že rozhodovací fonetický strom je v případě monofonových modelů velmi jednoduchý a obsahuje pouze základní větvení na úrovni hlásek. Kaldi ke všem řečovým jednotkám přistupuje jako ke

kontextově závislým a tak je tento strom vytvořen i v případě monofonových modelů. Standardně se rozhodovací strom vytváří v binární podobě, ale je možné jej převést do tvaru čitelném pro člověka pomocí nástroje `draw-tree`. Část takto převedeného rozhodovacího stromu můžeme vidět na obrázku 5.3.



Obrázek 5.3: Část rozhodovacího stromu pro monofonové modely.

Další částí při trénování monofonových modelů je nástroj `compile-train-graphs`. Tento nástroj sestaví pro každou promluvu odpovídající FST, které uloží v archivním formátu (.ark). Vstupem do tohoto nástroje je rozhodovací strom vytvořený v předchozím kroku (`tree`), inicializované HMM (`0.mdl`), slovník ve FST tvaru (`L.fst`) a transkripce řečových promluv přemapované do číselné podoby podle souboru `words.txt` (`rspecifier-promluv`). Poslední částí je specifikace výstupních souborů s výslednými FST řečových promluv (`wspecifier-grafy`).

compile-train-graphs `tree 0.mdl L.fst rspecifier-promluv wspecifier-grafy`

Strukturu grafů pro jednotlivé promluvy si lze zobrazit pomocí nástroje `fstcopy`, což můžeme vidět na obrázku 5.4.

Vstupními symboly jsou tzv. *transition-id*, které můžeme chápat jako identifikátory podobné *pdf-id*[7]. Jelikož Kaldi podporuje sdílení stejných *pdf-id* pro různé stavy, vznikají zde problémy při zpětném získání informací z těchto identifikátorů (například jaké hlásce patří dané *pdf-id*). *Transition-id* v sobě uchovává více informace, než *pdf-id* a řeší tak problém nejednoznačnosti.

Na obrázku 5.4 máme dva výstupní symboly a to 369 a 151. Co tato čísla znamenají se lze dozvědět ze souboru `words.txt`. Zde konkrétně je 369 – „se“ a 151 – „kilogramů“.

[7]Tento identifikátor byl přiblížen v části 3.3, kde byl pojmenován *pdf-class*. Oba dva tvary jsou mají stejný význam.

ze_stavu	do_stavu	vstupní_symbol	výstupní_symbol
274	93	326	369
275	690	35	0
276	277	18	0
277	78	20	0
277	277	21	0
278	691	22	0
279	385	326	369
279	386	54	0
279	387	36	0
279	78	284	151

Obrázek 5.4: Část grafu řečové promluvy dopln_00003_00 – _BREATH_ Kolik kilogramů se tedy dá reálně zhubnout za měsíc?

Aby mohlo být trénování monofonových modelů zahájeno, je použit nástroj `align-equal-compiled`. Ten pro každou promluvu rovnoměrně rozloží hranice modelů. Toto zarovnání, neboli „alignment“ je v Kaldi reprezentováno vektorem identifikátorů *transition-id*. Tento nástroj dále vypočítá různé statistiky ze získaného zarovnání modelů, které poté Kaldi používá při trénování modelů. Získání těchto statistik se provádí za pomoci nástroje `gmm-acc-stats-ali`.

Následně je provedena první aktualizace parametrů modelů pomocí nástroje `gmm-est`. Tento nástroj provádí reestimaci parametrů ve smyslu metody podle maximální věrohodnosti a používá se v následujícím tvaru:

gmm-est `0.mdl 0.acc 1.mdl`

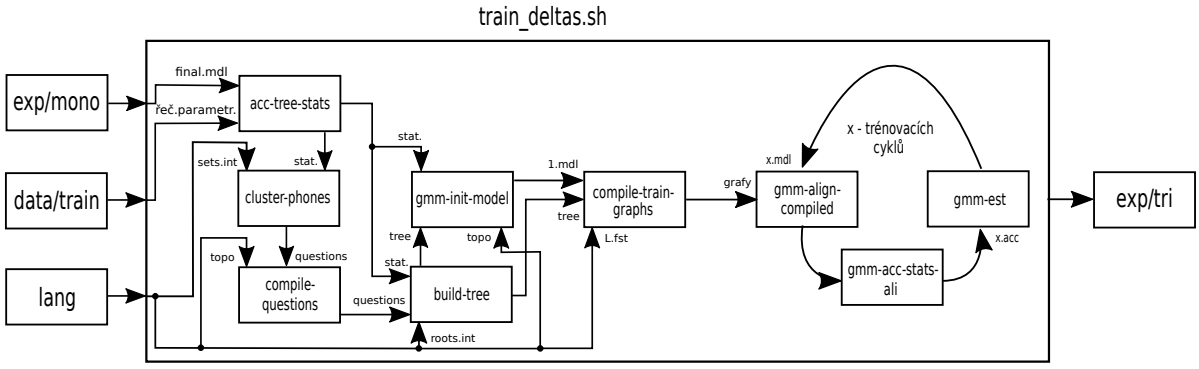
Vstupem jsou inicializované monofonové modely `0.mdl` společně se statistikami, které byli získány z předchozího kroku nástrojem `align-equal-compiled` (`0.acc`). Výstupem jsou nové, přesnější modely označené podle prvního iteračního kroku číslem 1 (`1.mdl`).

Následuje fáze zpřesňování parametrů HMM, která sestává z předem stanoveného počtu kroků. Počet těchto kroků je volně nastavitelný, ve standardním nastavení je to 40 kroků. První částí procesu zpřesňování parametrů HMM je zarovnání pomocí nástroje `gmm-align-compiled`. Zarovnání se však neprovádí v každém kroku, ale pouze v několika předem stanovených. Skript `train_mono.sh` toto zarovnání standardně provádí v krocích 1, 2, 3, 4, ,5 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 23, 26, 29, 32, 35 a 38. Toto nastavení jsme pro první experimenty rovněž ponechali. Po této fázi jsou opět vypočtené potřebné statistiky pro reestimaci nástrojem `gmm-acc-stats-ali` a parametry modelů jsou reestimovány nástrojem `gmm-est`. Tento proces, který můžeme názorně vidět na obrázku 5.2 se stále opakuje až do dosažení mezního počtu trénovacích kroků.

Před ukončením procesu trénování monofonových modelů se ještě provádí finální zarovnání podle monofonových modelů, použitím skriptu `align_si.sh`. Tento krok se provádí na všech datech řečového korpusu a s nejaktuálnějšími modely, tak aby bylo získáno nejpřesnější možné zarovnání modelů.

5.5.2 Trénování trifonových modelů

Po trénování monofonových modelů se přechází na trénování kontextově závislých modelů – trifonových modelů. Skript, který se pro tento účel používá se jmenuje `train_deltas.sh`. Stejně jako skript `train_mono.sh` v případě monofonových modelů, tak i `train_deltas.sh` v případě trifonových modelů se skládá z několika dílčích kroků. Jeho zjednodušenou vnitřní strukturu můžeme vidět na obrázku 5.5.



Obrázek 5.5: Vnitřní struktura nástroje `train_deltas.sh` používaném pro trénování trifonových modelů.

Tento skript se používá v následujícím tvaru:

train_deltas `$num_leaves $tot_gauss data/train lang exp/mono_ali exp/tri_delta`

První argument tohoto nástroje je `$num_leaves`, což je maximální počet listů u rozhodovacích fonetických stromů. Tato proměnná závisí na počtu dat, počtu hlásek v daném jazyku a na celkovém počtu *pdf-id*. Z analýzy Kaldi příkladů, ze kterých jsme vycházeli (především z [9], kde byl použit český jazyk a podobná velikost dat) se jako dostačující hodnota `$num_leaves`, pro naše účely ukázalo 1000 a tato hodnota byla použita i v nastavení našich experimentů. Dalším argumentem nástroje `train_deltas.sh` je `$tot_gauss`, což je celkový počet gaussovských směsí u hustotních funkcí jednotlivých stavů. Stejně jako v případě proměnné `$num_leaves` jsme vycházeli z analýzy volně dostupných příkladů a hodnota `$tot_gauss` byla nakonec nastavena na hodnotu 10000. Následujícími argumenty nástroje `train_deltas.sh` jsou složka s datovými soubory (`data/train`), složka s jazykovými specifikacemi (`lang`), složka s finálním zarovnáním pomocí monofonových modelů (`exp/mono_ali`) a výstupní složka pro natrénované trifonové modely (`exp/tri_delta`).

Jak už bylo popsáno v části 3.4.3, trifonové modely je potřeba shlukovat, což se v Kaldi provádí pomocí fonetických rozhodovacích stromů. V klasickém přístupu v HTK se rozhodovací stromy vytváří za pomoci předem daných otázek (nejčastěji dotazujících se na

okolní kontext dané hlásky), které jsou zpravidla určeny expertem. V Kaldi jsou však, v základním nastavení, tyto otázky získávány automaticky.

První krokem v tomto procesu je nástroj `acc-tree-stats`. Tento nástroj projde řečové parametrizace a společně se zarovnáním monofonových modelů vypočítá potřebné statistiky pro sestavení rozhodovacích stromů. Tyto statistiky většinou obsahují informace o akustické podobnosti jednotlivých hlásek na jejichž základě je Kaldi schopné automaticky vytvořit otázky. Nástroj se používá v následujícím tvaru :

acc-tree-stats `final.mdl $feats rspecifier-alignment treeacc`

Vstupem jsou řečové parametrizace (`$feats`), monofonové modely (`final.mdl`) a zarovnání podle monofonových modelů (`rspecifier-alignment`). Výstupem pak jsou statistiky pro sestavení rozhodovacích stromů (`treeacc`).

Další částí je nástroj `cluster-phones`. Ten ze získaných statistik automaticky vytvoří otázky. V této části jsou také automaticky přidány otázky, které byly definovány v souboru `extra_questions.txt`. V Kaldi se otázky mohou vyskytovat v několika formátech. Jedním z nich je například číselný formát, kde každé číslo označuje danou hlásku, podle souboru `phones.txt`. Jako příklad můžeme uvést část souboru `questions.int`, který obsahuje automaticky vytvořené otázky v jejich číselné reprezentaci. Tato číselná reprezentace je poté pro větší názornost převedena do textové formy podle souboru `phones.txt` (pro více informací lze nahlédnout do Přílohy B, kde je uvedena fonetická abeceda – EPA):

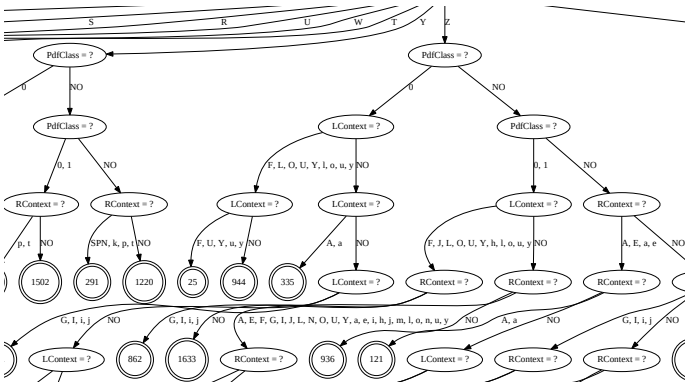
5	7	27	30		
9	22	25	39	45	49
10	12	34	37		
A	E	a	e		
F	U	Y	l	u	y
G	I	i	j		

Následně je sestaven rozhodovací strom pomocí nástroje `build--tree`, který je použit ve tvaru:

build-tree `treeacc $num_leaves questions.qst topo tree`

Argument `$num_leaves` označuje maximální počet listů v rozhodovacím stromu. Vstupem jsou statistiky pro sestavení rozhodovacích stromů (`treeacc`), automaticky vytvořené otázky (`questions.qst`) a topologie HMM (`topo`). Výstupem je rozhodovací strom (`tree`), který je stejně jako v případě monofonových modelů v binárním formátu. Tento binární strom je možné opět převést do formátu čitelném pro člověka, pomocí nástroje `draw-tree`, což můžeme vidět na obrázku 5.6.

Další částí skriptu `train_deltas.sh` je inicializace trifonových modelů. Tento krok je proveden nástrojem `gmm-init-model`, který je použit ve tvaru:



Obrázek 5.6: Část rozhodovacího stromu pro trifonové modely.

gmm-init-model `tree treeacc topo 1.mdl`

Vstupem je binární rozhodovací strom (`tree`), statistiky tohoto rozhodovacího stromu (`treeacc`) a topologie modelů HMM (`topo`). Výstupem jsou pak inicializované trifonové modely (`1.mdl`).

Další částí je nástroj `compile-train-graphs`, který byl použit už při trénování monofonových modelů. V případě trifonových modelů se používá ve tvaru:

compile-train-graphs `tree 1.mdl L.fst rspecifier-promluv wspecifier-grafy`

Vstupem je zde binární rozhodovací strom (`tree`), inicializované trifonové modely (`1.mdl`), slovník ve tvaru FST (`L.fst`) a přepis řečových promluv přemapovaný do číselné podoby podle souboru `words.txt` (`rspecifier-promluv`). Výstupem jsou FST řečových promluv (`wspecifier-grafy`).

Jako poslední se provádí část zpřesňování parametrů trifonových HMM, která probíhá stejně jako v případě monofonových HMM. Opět se zde opakovaně používají nástroje `gmm-align-compiled`, `gmm-acc-stats-ali` a `gmm-est`, které byly popsány již v případě trénování monofonových HMM (v části 5.5.1). Rozdíl je zde v počtu trénovacích kroků, jejichž počet byl snížen na 35 a zarovnání se zde provádí pouze v krocích 10, 20 a 30. Finální zarovnání je opět provedeno pomocí nástroje `align_si`, stejně jako v případě trénování monofonových modelů.

5.6 Úpravy výsledné segmentace

V této části uvedeme úpravy, které bylo nutné provést, aby mohli být výsledky z Kaldi vyhodnoceny stejným způsobem, jako v HTK. Všechny kroky, které zde budou popsány jsou součástí skriptů ze složky `local`.

5.6.1 Převod souboru CTM do MLF

Zarovnání modelů (dále již označované jako segmentace), které jsme získali v předchozím kroku, lze převést do formátu CTM pomocí nástroje `ali-to-phones`. Tento formát je popsán například v [3] a má následující tvar:

$$\boxed{<NP><KP><ZH><TH>\ OH}$$

V tomto souboru je tak pro každou hlásku dané řečové promluvy specifikováno:

- Název řečové promluvy, ze které hláska pochází (NP)
- Kanál řečové promluvy, ze které hláska pochází (KP)
- Začátek hlásky určený od začátku řečové promluvy [s] (ZH)
- Trvání hlásky [s] (TH)
- Označení hlásky (OH), které je zde v číselné podobě (podle souboru *phones.txt*)

Jako příklad je zde uvedena část souboru CTM pro řečovou promluvu
`dopln000020_00` _BREATH_ *Co zbývá?*

dopln00020_00	1	0.00	0.64	4
dopln00020_00	1	0.64	0.22	29
dopln00020_00	1	0.86	0.19	41
dopln00020_00	1	1.05	0.19	52
dopln00020_00	1	1.24	0.17	30
dopln00020_00	1	1.41	0.41	13
dopln00020_00	1	1.82	0.12	49
dopln00020_00	1	1.94	0.75	6
dopln00020_00	1	2.69	0.13	1
dopln00020_00	1	2.82	0.56	2

Aby bylo možné vyhodnotit výsledky segmentace stejným způsobem jako v HTK, je soubor ve formátu CTM převeden do formátu MLF (z anglického spojení Master Label File). Tento soubor může v HTK sloužit k mnoha účelům a může obsahovat různé informace, zde je však použit pouze v následujícím tvaru:

$$\boxed{\begin{array}{l} \#!MLF!\# \\ \textit{název promluvy} \\ <\text{ ZH }><\text{ KH }>\text{ OH} \\ \vdots \\ \textit{název promluvy} \\ <\text{ ZH }><\text{ KH }>\text{ OH} \end{array}}$$

Název promluvy zde není definován na každém řádku, ale pouze na začátku promluvy a pro každou hlásku každé promluvy je zde specifikováno:

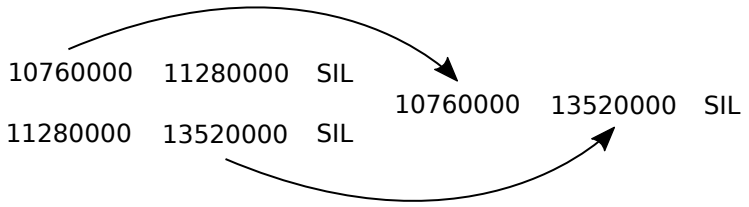
- Začátek hlásky počítaný od počátku řečové promluvy [ms] (ZH)
- Konec hlásky počítaný od počátku řečové promluvy [ms] (KH)
- Označení hlásky (OH), které je zde v textové formě.

Jako příklad je zde uvedena část souboru MLF pro řečovou promluvu `dopln000020_00` _BREATH_ *Co zbývá?*

"*/dopln00020_00.rec"		
0	2560000	SIL
2560000	3440000	c
3440000	4200000	o
4200000	4960000	z
4960000	5640000	b
5640000	7279999	I
7280000	7760000	v
7760000	10760000	A
10760000	11280000	SIL
11280000	13520000	SIL

5.6.2 Svázání pauz

Jednou z úprav výsledné segmentace je svázání pauz. Pro tuto část byl vytvořen skript `modif_Kaldi_mlf.sh`, který využívá přístup svázání pauz, který byl použit v HTK (konkrétně skript `mlfedit.py` ze složky `bin`). Tento skript se nachází ve složce `local`. Pauzy, které se nachází těsně vedle sebe (většinou na začátku a na konci věty) jsou sdruženy do jedné. Čas začátku první pauzy je čas začátku sdružené pauzy a čas konce poslední pauzy je čas konce sdružené pauzy, což můžeme vidět na obrázku 5.7.



Obrázek 5.7: Svázání pauz v souboru MLF.

5.6.3 Posun časových hranic hlásek

Další úpravou výsledné segmentace je posunutí časových hranic hlásek o předem daný offset. Tento proces je rovněž prováděn skriptem `mlfedit.py` (ze složky `bin`), volaného skriptem `modif_Kaldi_mlf.sh` (ze složky `local`). Důvody tohoto kroku jsou podrobně popsány v [12], kde bylo zjištěno, že v HTK dochází, při parametrizaci, k posunu hranic vypočtených vektorů parametrů. Tento posun se posléze projeví na přesnosti segmentace. Proto se do algoritmu zahrnuje offset (S), který je vypočítán podle vzorce:

$$S = \frac{L_W - L_F}{2} \quad . \tag{5.1}$$

L_W označuje délku okénka použitého při parametrizaci řečových promluv a L_F označuje posun tohoto okénka. Pokud tedy budeme uvažovat délku okénka (L_W) 25 ms a posun tohoto okénka (L_F) 6 ms, bude mít posun (S) hodnotu 9,5 ms, tj. k výsledným časovým hranicím se připočte 9,5 ms (posunou se o 9,5 ms „doprava“). Tato hodnota offsetu je použita v základním nastavení segmentace v této práci. Experimentální vyhodnocení zda má tento offset vliv na přesnost časových hranic hlásek i v Kaldi je provedeno v části 6.

5.7 Modifikace automatické segmentace v Kaldi

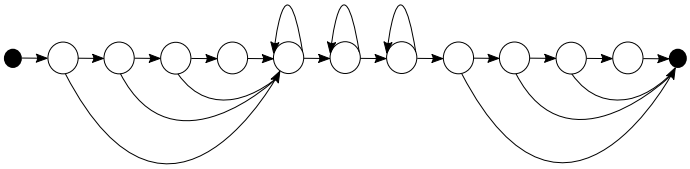
5.7.1 Upravená topologie 5-ti stavového modelu

Hlavní modifikací původního algoritmu je implementace nové topologie HMM, kterou jsme si přiblížili v části 3.3. Tato topologie vychází z původní topologie klasického 5-ti stavového modelu HMM, ale používá více krajních stavů. Do algoritmu automatické segmentace je implementována pomocí souboru `topo`. Na obrázku 5.8 můžeme vidět, jak tento soubor vypadá společně s grafickým znázorněním použité topologie.

V definici této topologie můžeme vidět, že krajní stavy sdílí stejné *pdf-id*, což znamená, že při trénování svých parametrů budou sdílet stejnou hustotní funkcí. To je v souladu s definicí, že krajní stavy rozšířeného modelu jsou kopie krajního stavu v původním 5–ti stavovém HMM.

5.7.2 Krátká pauza

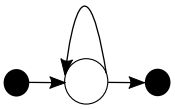
V HTK se používají dva druhy pauz – dlouhá (SIL) a krátká (SP), jak už bylo popsáno v části 5.3. V HTK je krátká pauza automaticky vložena mezi každá dvě slova a pokud se v průběhu segmentace zjistí, že mezi danými slovy není, tak se jí nastaví nulová délka. V Kaldi tento způsob nemůže být použit, což bylo rovněž popsáno v části 5.3. Zvolili jsme zde tedy odlišný způsob reprezentace krátké pauzy, který můžeme vidět na obrázku 5.9. Tento model je podobný modelu krátké pauzy z HTK, ovšem nedovoluje kompletní přeskočení modelu. Tato možnost je implementována tak, že ve slovníku je pro každé slovo navíc zavedena fonetická transkripce tohoto slova i s krátkou pauzou. Viterbiův algorit-



```
<TopologyEntry>
<ForPhones>
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 1 0.5 <Transition> 4 0.5 </State>
<State> 1 <PdfClass> 0 <Transition> 2 0.5 <Transition> 4 0.5 </State>
<State> 2 <PdfClass> 0 <Transition> 3 0.5 <Transition> 4 0.5 </State>
<State> 3 <PdfClass> 0 <Transition> 4 1 </State>
<State> 4 <PdfClass> 1 <Transition> 4 0.5 <Transition> 5 0.5 </State>
<State> 5 <PdfClass> 2 <Transition> 5 0.5 <Transition> 6 0.5 </State>
<State> 6 <PdfClass> 3 <Transition> 6 0.5 <Transition> 7 0.5 </State>
<State> 7 <PdfClass> 4 <Transition> 8 0.5 <Transition> 11 0.5 </State>
<State> 8 <PdfClass> 4 <Transition> 9 0.5 <Transition> 11 0.5 </State>
<State> 9 <PdfClass> 4 <Transition> 10 0.5 <Transition> 11 0.5 </State>
<State> 10 <PdfClass> 4 <Transition> 11 1 </State>
<State> 11 </State>
</TopologyEntry>
```

Obrázek 5.8: Upravená topologie 5-ti stavového HMM modelu se čtyřmi rozšiřujícími krajními stavy a její reprezentace v Kaldi.

mus poté při segmentaci vybere možnost, která bude poskytovat lepší výsledky, tedy buď s pauzou, nebo bez pauzy.



```
<TopologyEntry>
<ForPhones>
1
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.5 <Transition> 1 0.5 </State>
<State> 1 </State>
</TopologyEntry>
</Topology>
```

Obrázek 5.9: Topologie krátké pauzy použité v Kaldi.

5.7.3 Expertně vytvořené otázky

V Kaldi se otázky pro vytvoření fonetických rozhodovacích stromů vytvářejí automaticky, což bylo uvedeno už v části 5.5.2. V HTK se ovšem používají otázky předem vytvořené expertem. Ten vytvoří otázky jako posloupnost řečových jednotek, které jsou navzájem

podobné (například na základě akustické podobnosti). V této práci rovněž zkoumáme vliv použití těchto předem vytvořených otázek na přesnost automatické segmentace v Kaldi. Expertně vytvořené otázky byly uloženy do souboru *extra_questions.txt*, který byl popsán v části 5.2.2. Jako příklad uvádíme část tohoto souboru.

HTK					
DlouhV	A	E	I	O	U
PredV	e	i	E	I	
ZadV	o	u	O	U	

Kaldi.txt				
A	E	I	O	U
e	i	E	I	
o	u	O	U	

Kaldi.int				
6	8	13	18	23
31	35	8	13	
41	46	18	23	

5.7.4 Počet Gaussovských směsí

Další úprava je zaměřena na počet Gaussovských směsí ve výstupní hustotní funkci stavů modelů u monofonových a trifonových modelů. Jak je ukázáno v [13], pokud je k dispozici velké množství dat od jednoho řečníka, nejlepších výsledků automatické segmentace je dosaženo pro:

- 1 Gaussovskou směs v případě výstupní hustotní funkce trifonových modelů.
- 4-8 Gaussovských směsí v případě výstupní hustotní funkce monofonových modelů .

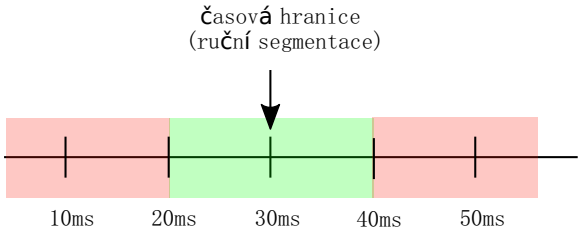
Zvyšování počtu Gaussovských směsí následně (podle [13]) vede ke zhoršení přesnosti segmentace. V části 6.3.3 je experimentálně vyhodnocen vliv tohoto nastavení na algoritmus automatické segmentace v Kaldi.

Kapitola 6

Vyhodnocení výsledků

6.1 Způsob vyhodnocení

K dispozici máme přibližně 50 ručně nasegmentovaných promluv. Výsledky automatické segmentace jsou následně vyhodnoceny pomocí odchylky od ručních segmentací na zvolenému tolerančnímu pásmu. Pokud je odchylka automatické segmentace od ruční segmentace větší, než zvolené pásmo, je vyhodnocena jako chybná. Pokud se výsledek automatické segmentace nachází uvnitř tolerančního pásma, je tento výsledek vyhodnocen jako správný.



Obrázek 6.1: Příklad vyhodnocení přesnosti automatické segmentace s použitím tolerančního pásma 10 ms (červená oblast - výsledek vyhodnocen jako chybný, zelená oblast - výsledek vyhodnocen jako správný).

6.2 Porovnání – základní algoritmus Kaldi s HTK

V této části je porovnáno základní nastavení automatické segmentace v Kaldi s automatickou segmentací v HTK. Zatímco nastavení segmentace v Kaldi je z velké části sestaveno z nejčastějších parametrů získaných z volně dostupných příkladů Kaldi, nastavení segmentace v HTK je již odladěno tak, aby poskytovalo co nejlepší výsledky. Nyní si zopakujeme nejdůležitější nastavení, společně s nastavením použitým v HTK:

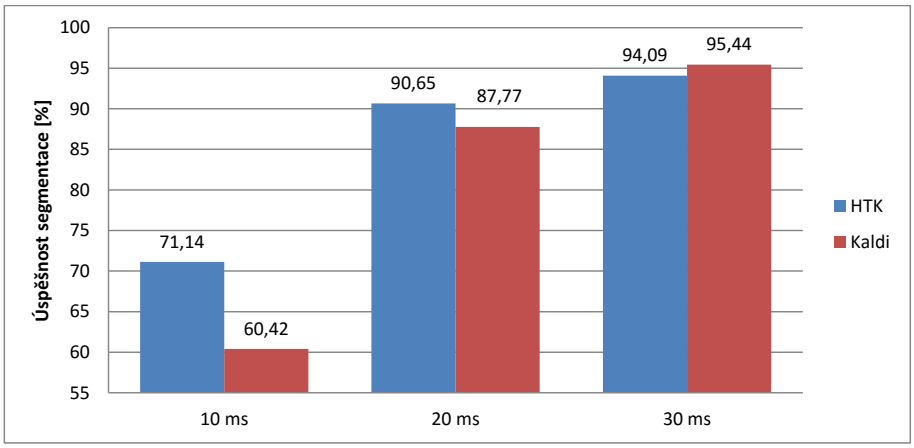
Kaldi

- Parametry řečových promluv – MFCC parametry získané pomocí Hammingova okénka s délkou 25 ms a posunem 6 ms.
- Pauzy – Pauza SIL, kterou používá Kaldi (pro přehlednost, zde bude značena jako SIL_K) (popsána v části 5.3, obrázek 5.1b).
- Model – Klasický 5–ti stavový model (popsaný v části 5.3, obrázek 5.1a).
- Otázky pro vytvoření rozhodovacích stromů – Pouze automaticky vytvořené otázky.
- Počet Gaussovských směsí – V případě monofonových modelů je nastaveno 1000 Gaussovských směsí, v případě trifonových modelů je nastaveno 10000 Gaussovských směsí.
- Použit offset.

HTK

- Parametry řečových promluv – MFCC parametry získané pomocí Hammingova okénka s délkou 25 ms a posunem 6 ms.
- Pauzy – Krátká pauza (SP) a dlouhá pauza (SIL) (popsány v části 3.3, obrázek 3.4).
- Model – Klasický 5–ti stavový model (popsaný v části 5.3, obrázek 5.1).
- Otázky pro vytvoření rozhodovacích stromů – Pouze expertně vytvořené otázky.
- Počet Gaussovských směsí – V případě monofonových modelů jsou nastaveny 4 Gaussovské směsi a v případě trifonových modelů je nastavena 1 Gaussovská směs.
- Použit offset.

Jak můžeme vidět z obrázku 6.2, základní algoritmus v Kaldi dosahuje podobných výsledků jako algoritmus v HTK. Na největším tolerančním pásmu(30 ms) je přesnost segmentace za pomoci Kaldi dokonce lepší. Naopak na nejmenším tolerančním pásmu (10 ms) je úspěšnost segmentace za pomoci HTK výrazně lepší.



Obrázek 6.2: Srovnání základního algoritmu v Kaldi s algoritmem používaným v HTK. Vyhodnocení je provedeno na třech různých tolerančních pásmech (10, 20 a 30 ms).

6.3 Vyhodnocení – úpravy segmentace v Kaldi

V této části jsou parametry segmentace v Kaldi nastaveny na stejné hodnoty, jako při použití nástroje HTK. V částech, u kterých není možné použít stejné nastavení jako v HTK jsou experimentálně vyzkoušeny nejrůznější varianty nastavení a je zkoumána úspěšnost segmentace při jejich použití. Dále je vyhodnocen vliv jednotlivých nastavení z HTK na přesnost segmentace v Kaldi. Nastavení parametrů automatické segmentace v Kaldi je v této části následující:

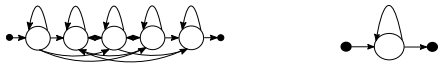
- Parametry řečových promluv – MFCC parametry získané pomocí Hammingova okénka s délkou 25 ms a posunem 6 ms.
- Pauzy – Model dlouhé (SIL – bez přechodu mezi stavy 2 a 4) a krátké pauzy (SP – bez možnosti přeskočení modelu) (důvod tohoto nastavení je uveden v části 6.3.1).
- Model – Klasický 5–ti stavový model (popsaný v části 5.3, obrázek 5.1a).
- Otázky pro vytvoření rozhodovacích stromů – Expertně vytvořené otázky (více v části 6.3.2).
- Počet Gaussovských směsí – V případě monofonových modelů jsou nastaveny 4 Gaussovské směsi a v případě trifonových modelů je nastavena 1 Gaussovská směs (více v části 6.3.3).
- Použit offset (více v části 6.3.4).

6.3.1 Modely pauz

V této části je prozkoumán vliv různých HMM pauz na celkovou přesnost automatické segmentace v Kaldi. Jelikož zde není možné použít stejné pauzy, jaké byly použity v HTK, je zde vyzkoušeno několik HMM pauz a jejich kombinací.

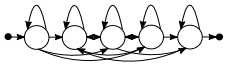
- **SIL + SP**

- **SIL_K + SP**



- **SIL_HTK + SP**

- **SIL_K**



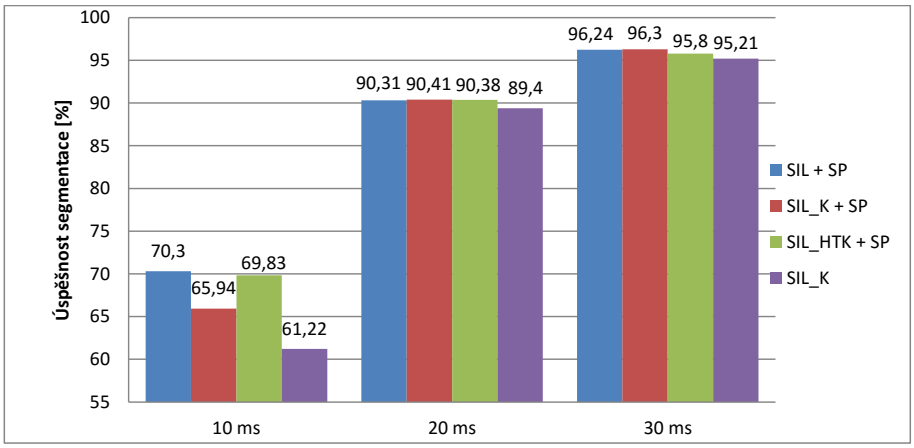
Na obrázku 6.3 můžeme vidět vyhodnocení jednotlivých kombinací HMM pauz. Pro modelování pauz se, jako nejlepší výsledek, ukázalo použití kombinace SIL+SP, u které se přesnost segmentace zvýšila především na nejmenším tolerančním pásmu (10 ms).

6.3.2 Otázky pro generování rozhodovacích stromů

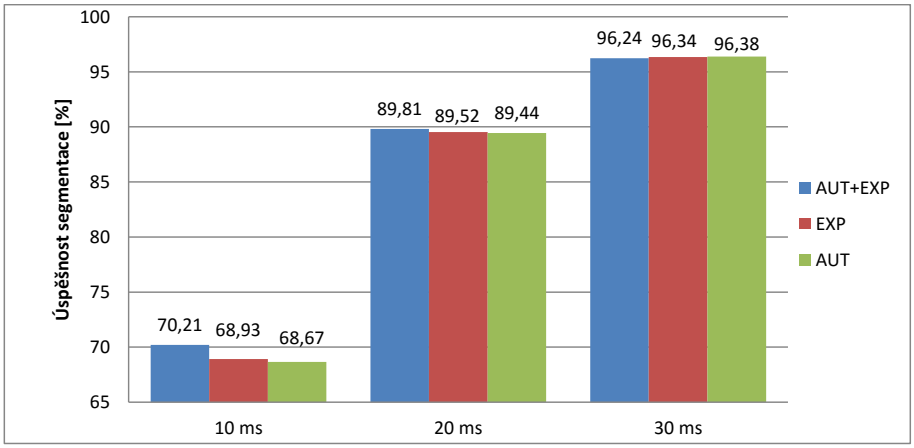
V této části byl vyhodnocen vliv způsobu generování otázek na přesnost celkové segmentace. Tato problematika byla popsána v části 5.5.2 a 5.7.3. Pro vyhodnocení bylo použito následujících nastavení:

- Otázky předem definované expertem (EXP)
- Otázky automaticky vygenerované systémem Kaldi (AUT)
- Kombinace obou možností (EXP+AUT)

Jak můžeme vidět na obrázku 6.4, při použití automatických otázek vytvořených pomocí Kaldi je dosaženo téměř totožných výsledků, jako při použití expertních otázek. Zároveň bylo zjištěno, že přesnost segmentace se ještě zvýší, pokud je použito expertních i automaticky vygenerovaných otázek.



Obrázek 6.3: Kombinace různých HMM pauz a ověření přesnosti segmentace v Kaldi při jejich použití.



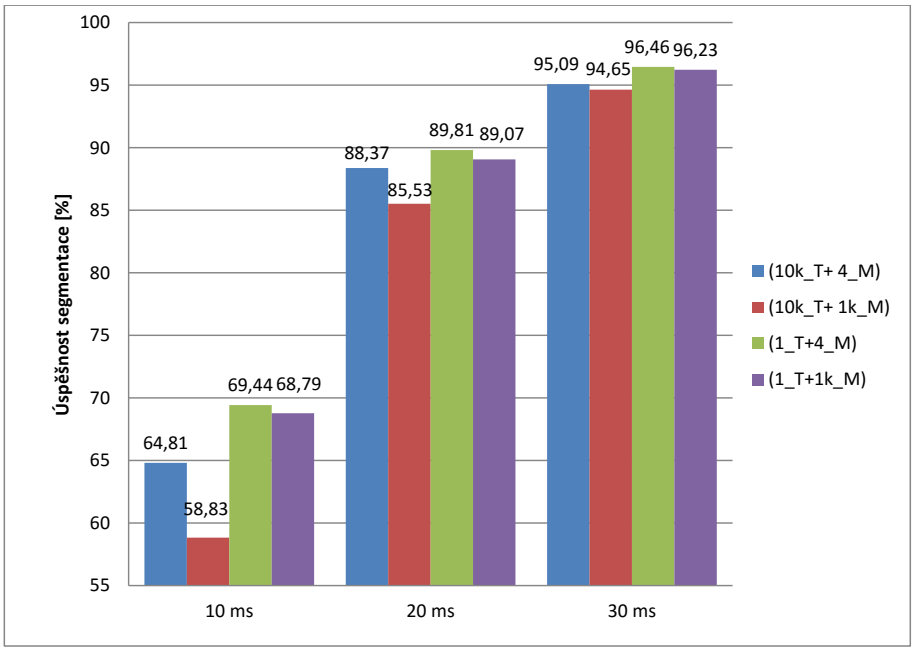
Obrázek 6.4: Vyhodnocení přesnosti automatické segmentace v Kaldi při použití různých způsobů generování otázek.

6.3.3 Počet Gaussovských směsí

V této části je experimentálně vyhodnocen vliv počtu Gaussovských směsí ve výstupních hustotních funkcích stavů modelu, jak u monofonových, tak u trifonových modelů (jak bylo uvedeno v části 5.7.4). Vyhodnocení je provedeno pro kombinace následujících nastavení:

- 10000 Gaussovských směsí ve výstupní hustotní funkci stavů **trifonových** modelů (10k_T)
- 1 Gaussovská směs ve výstupní hustotní funkci stavů **trifonových** modelů (1_T)
- 1000 Gaussovských směsí ve výstupní hustotní funkci stavů **monofonových** modelů (1k_M)

- 4 Gaussovské směsi ve výstupní hustotní funkci stavů **monofonových** modelů (4_M)



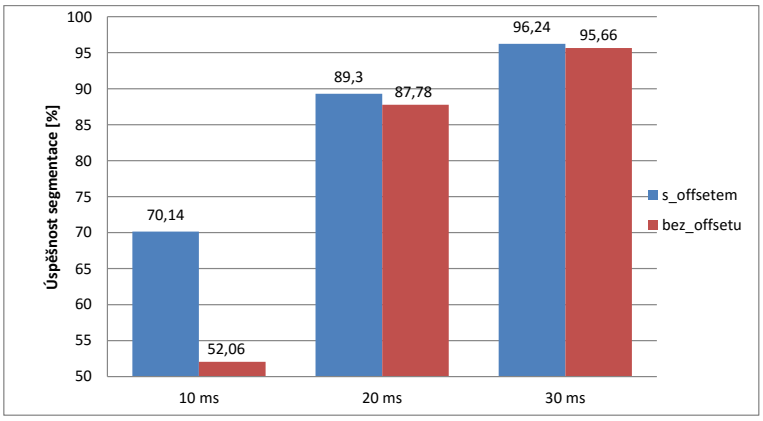
Obrázek 6.5: Porovnání přesnosti automatické segmentace při použití různého počtu Gaussovských směsí ve výstupní hustotní funkci stavů modelů.

Z obrázku 6.5 vyplývá, že zvyšování počtu Gaussovských směsí ve výstupní hustotní funkci stavů modelů opravdu způsobí snížení přesnosti výsledné segmentace a to především v případě trifonových modelů. Nejlepších výsledků bylo dosaženo pro 1 Gaussovskou směs pro trifonové modely a 4 Gaussovské směsi pro monofonové modely (stejně jako ve studii, kterou jsme se zmiňovali v části 5.7.4).

6.3.4 Offset

Zde je vyhodnoceno použití offsetu, neboli posunutí výsledné segmentace o určitou hodnotu. Tento offset byl popsán v části 5.6.3.

Jak můžeme vidět z obrázku 6.6, použití offsetu vede k výrazně lepším výsledkům. To potvrzuje, že v Kaldi dochází ke stejnému posunu vektorů parametrů jako v HTK.



Obrázek 6.6: Porovnání přesnosti automatické segmentace při použití offsetu a bez použití offsetu.

6.4 Porovnání – vylepšený algoritmus Kaldi s HTK

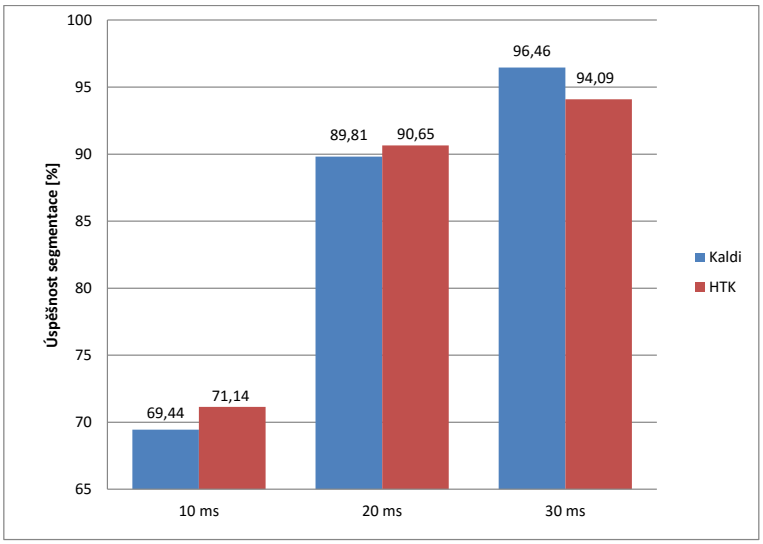
V této části je porovnán algoritmus automatické segmentace s použitím nejlepšího nastavení, které bylo experimentálně získáno v předchozí části. Jak můžeme vidět na obrázku 6.7, úspěšnost vylepšené segmentace v Kaldi je již téměř srovnatelná se segmentací v HTK.

6.5 Modifikovaný model HMM

V této části je vyhodnoceno použití modifikovaného HMM modelu pro modelování hlásek. Tento model byl popsaný v části 3.3 a 5.7. Na obrázku 6.8 můžeme vidět porovnání obou modelů, ze kterého vyplývá, že použití modifikovaného modelu vede ke zvýšení přesnosti segmentace, jak u nástroje Kaldi, tak u nástroje HTK.

6.6 Porovnání různých vektorů parametrů - MFCC a PLP

V této části je uvedeno porovnání řečových parametrizací MFCC a PLP při použití v algoritmu automatické segmentace v Kaldi. Vektory parametrů jsou vypočítány pomocí Hammingova okénka o délce 25 ms, pro posuny (4, 6, 8 a 10 ms). Výsledky jsou vyhodnoceny



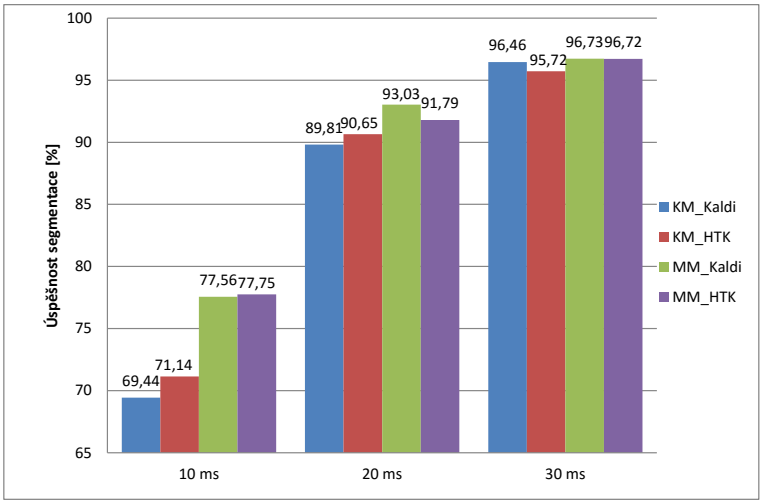
Obrázek 6.7: Porovnání vylepšené segmentace v Kaldi se segmentací v HTK

jak pro klasický, tak pro modifikovaný model. Z obrázků 6.9, 6.10, 6.11 a 6.12 vyplývá, že **nejlepších** výsledků je dosaženo:

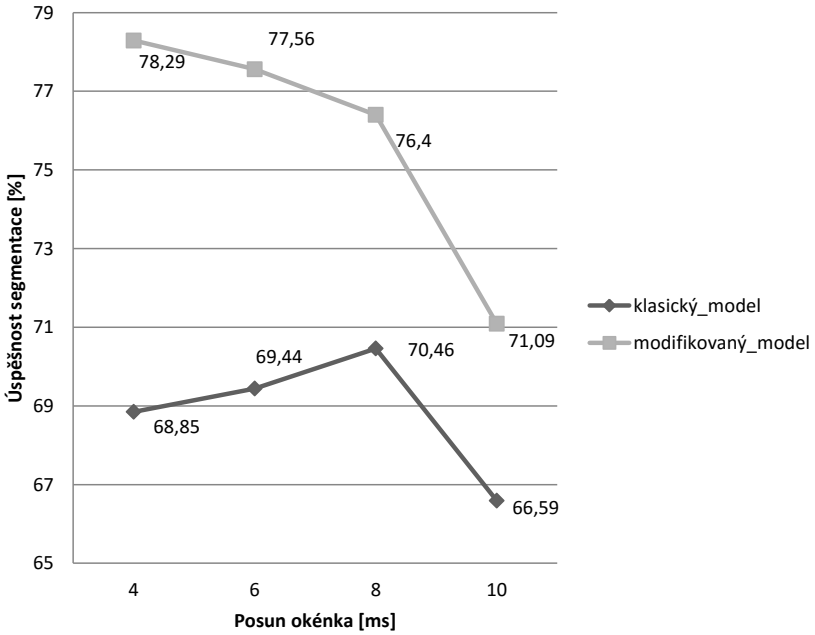
- pro toleranční pásmo : **10 ms**
  - vektory parametrů **PLP** s okénkem (**25/4**), při použití **modifikovaného** modelu. Úspěšnost automatické segmentace : **79,45 %**
- pro toleranční pásmo : **20 ms**
  - vektory parametrů **MFCC** s okénkem (**25/8**), při použití **modifikovaného** modelu. Úspěšnost automatické segmentace : **93,9 %**

a **nejhorších** výsledků je dosaženo:

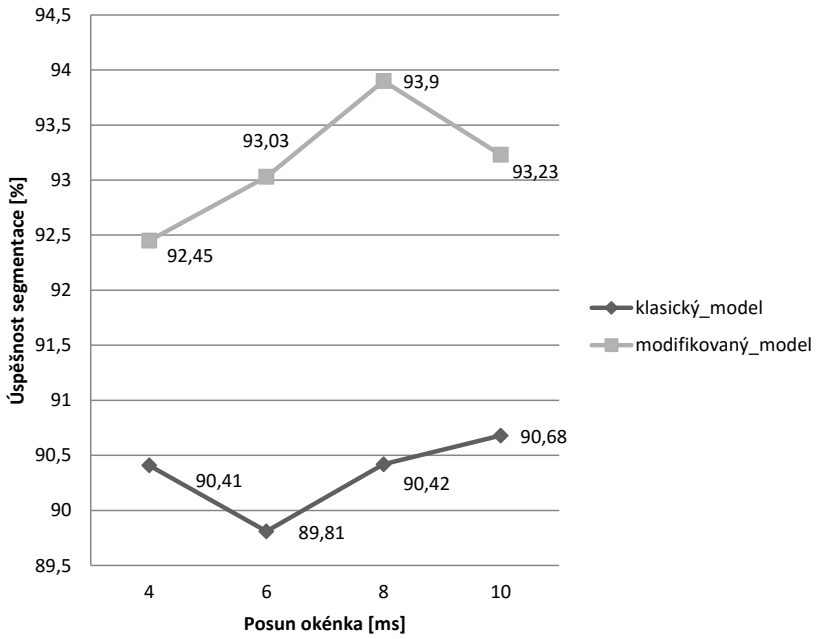
- pro toleranční pásmo : **10 ms**
  - vektory parametrů **MFCC** s okénkem (**25/10**), při použití **klasického** modelu. Úspěšnost automatické segmentace : **66,59 %**
- pro toleranční pásmo : **20 ms**
  - vektory parametrů **MFCC** s okénkem (**25/6**), při použití **klasického** modelu. Úspěšnost automatické segmentace : **89,81 %**



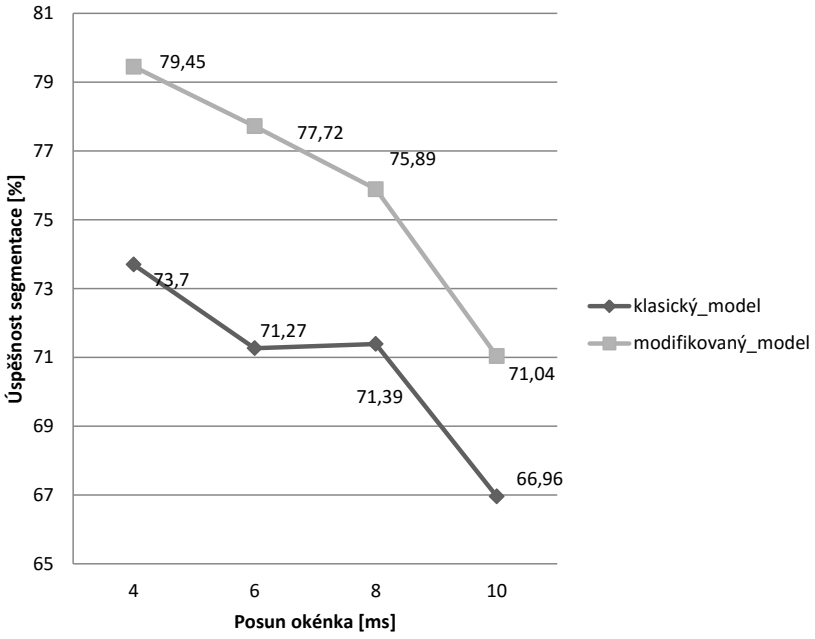
Obrázek 6.8: Porovnání klasického (KM) a modifikovaného modelu (MM) v obou nástrojích.



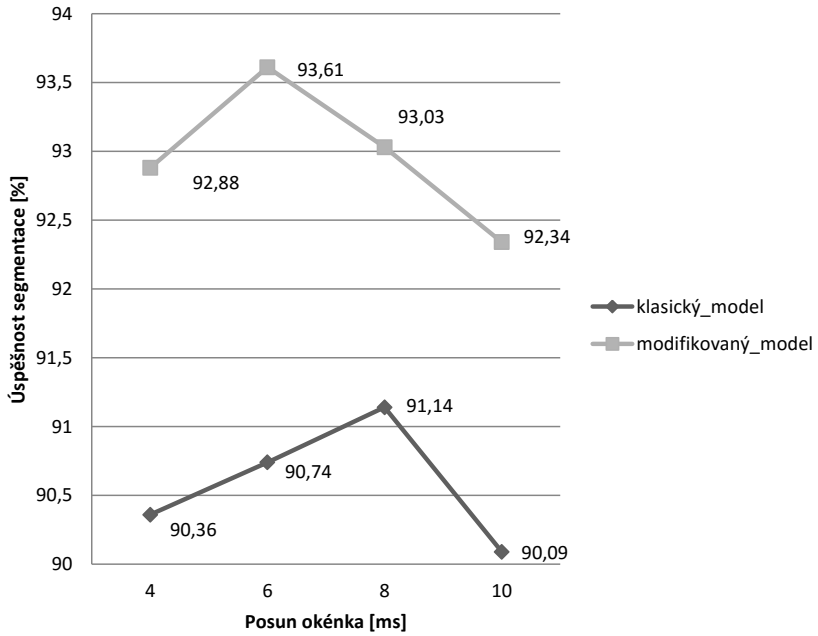
Obrázek 6.9: Porovnání klasického a modifikovaného modelu, při použití vektorů parametrů MFCC (délka okénka 25 ms s posuny 4, 6, 8 a 10 ms) na tolerančním pásmu 10 ms.



Obrázek 6.10: Porovnání klasického a modifikovaného modelu, při použití vektorů parametrů MFCC (délka okénka 25 ms s posuny 4, 6, 8 a 10 ms) na tolerančním pásmu 20 ms.



Obrázek 6.11: Porovnání klasického a modifikovaného modelu, při použití vektorů parametrů PLP (délka okénka 25 ms s posuny 4, 6, 8 a 10 ms) na tolerančním pásmu 10 ms.



Obrázek 6.12: Porovnání klasického a modifikovaného modelu, při použití vektorů parametrů PLP (délka okénka 25 ms s posuny 4, 6, 8 a 10 ms) na tolerančním pásmu 20 ms.

Kapitola 7

Závěr

Předkládaná práce se zabývá vytvořením automatické fonetické segmentace pomocí nástroje Kaldi s následným vylepšováním její přesnosti. Tato práce byla rozdělena do dvou částí. První částí bylo vytvoření automatické fonetické segmentace pomocí nástroje Kaldi, s využitím algoritmů a postupů, které byly použity v téže úloze, ale s použitím nástroje HTK. Jak zde bylo popsáno, postup použitý v HTK se nedá exaktně použít i v Kaldi. Proto byl nejdříve vytvořen nový algoritmus, který je založen na tradičních přístupech, které se používají v Kaldi. Jak můžeme vidět z výsledků (část 6.2), tento algoritmus nedělá velké chyby (úspěšnost segmentace na větších tolerančních pásmech je prakticky stejně přesná jako v HTK), ovšem při porovnání na menších tolerančních pásmech je výrazně horší. Následně bylo navrženo několik úprav této segmentace, které se zaměřili na analýzu postupů segmentace v HTK a možnosti přenesení těchto postupů do algoritmu segmentace v Kaldi. Bylo nalezeno několik úprav (offset, počty Gaussovských směsí), které přinesli zvýšení přesnosti segmentace i v nástroji Kaldi. Vyhodnocení této vylepšené segmentace bylo provedeno v části 6.3, kde již bylo dosaženo velmi podobných výsledků, jakých bylo dosaženo při segmentaci pomocí HTK.

Druhou částí této práce bylo použití modifikovaného HMM pro modelování hlásek a vyhodnocení jeho vlivu na celkovou přesnost segmentace. Toto vyhodnocení bylo provedeno v části 6.5, kde bylo zjištěno, že segmentace, při použití tohoto modifikovaného modelu, dosahuje lepších výsledků a to řádově o několik procent. Zvýšení přesnosti segmentace při použití modifikovaného modelu bylo prokázáno, jak v případě segmentace s využitím nástroje HTK, tak v případě segmentace s využitím nástroje Kaldi. Dále byly porovnány různé druhy řečových parametrizací, konkrétně MFCC a PLP, pro něž bylo zjištěno, že dosahují podobné přesnosti segmentace. Ovšem nejlepšího výsledku dosáhla PLP parametrizace.

V této práci jsme také zmiňovali jednoznačný vliv inicializace modelů pomocí bootstrapu na přesnost segmentace při použití nástroje HTK. Lze předpokládat, že by incializace modelů bootstrapem zvýšila přesnost segmentace i při použití Kaldi. Tento způsob inicializace však v době vytváření této práce ještě nebyl v Kaldi implementován a je tedy dobrým námětem pro budoucí práci. V navazující práci se rovněž nabízí možnost využití neuronových sítí, které mohou být použity pro potřeby automatické segmentace,

a se kterými Kaldi dokáže pracovat. Rovněž by bylo vhodné nadále zkoumat různé topologie modelů, neboť jak se v této práci ukázalo, ani tato oblast se nezdá být ještě zcela vyčerpána.

Kapitola 8

Přílohy

A Obsah přiloženého DVD

- **Diplom_prace** – hlavní složka se všemi algoritmy, které byly v této práci popsány.
  - **Anotace_a_slovnik** – obsahuje přepis všech promluv, při běhu programu se zde vytvoří slovník, případně vyfiltrované promluvy.
  - **configs** – soubor s konfiguračními soubory.
  - **HTK** – složka se soubory z původního návrhu v HTK, které se používají pro vyhodnocení výsledků.
  - **lang** – složka s jazykovými specifikacemi.
  - **local** – složka s programy, které byly vytvořeny přímo v této práci, k nejrůznějším účelům.
  - **lst a nlp** – složky, které jsou používány při vyhodnocení výsledků.
  - **parametry** – složka s uloženými parametry (MFCC a PLP), z důvodu jejich větší velikosti jsou zde uloženy jen pro okénka – 25/8 a 25/6.
  - **steps, utils** – jsou složky s Kaldi skripty.
  - **topologie** – složka s uloženými topologiemi, pro klasický a modifikovaný model.
  - **run.sh** – hlavní skript pro spuštění programu.
  - **path.sh** – skript pro nastavení potřebných cest.
  - **setvar.sh** – skript pro nastavení používaných proměnných.
- **Navod_pro_spusteni.pdf** – textový soubor, který obsahuje nutné informace pro správné spuštění programu.
- **DP.pdf** – diplomová práce ve formátu PDF.

B Fonetická abeceda

EPA	SAMPA	Příklad
i	i	p**i**vo
e	e	p**e**s
a	a	mám**a**
o	o	b**o**k
u	u	r**u**m
I	i:	v**í**no
E	e:	l**é**pe
A	a:	t**á**ta
O	o:	j**ó**d
U	u:	r**ů**že
y	o_u	p**ou**to
Y	a_u	**au**to
F	e_u	**eu**nuch
@	@	*hláskování*
p	p	**p**rak
b	b	**b**od
t	t	o**t**ec
d	d	**d**ům
T	c	ku**t**il
D	J\	**ď**ěti
k	k	o**k**o
g	g	**g**uma
!	?	*ráz*
	t_s	e**l**

EPA	SAMPA	Příklad
c		o**c**
w	d_z	le**c**kdo
C	t_S	o**č**i
W	d_Z	lé**č**ba
f	f	**f**acka
v	v	**v**lak
s	s	o**s**el
z	z	ko**z**a
R	P\	mo**ř**e
Q	Q\	**tř**i
S	S	po**š**ta
Z	Z	**ž**ena
j	j	vo**j**ák
x	x	**ch**yba
h	h\	**h**ad
G	G	aby**ch** byl
r	r	**r**ak
l	l	**l**oď
m	m	**m**ír
n	n	**n**os
N	N	ba**n**ka
J	J	la**ň**
M	F	ny**m**fa
L	l=	v**l**k

Seznam obrázků

Seznam zkratek

HTK	Hidden Markov Model Toolkit
HMM	skrytý Markovův model (z angličtiny – Hidden Markov Model)
MFCC	Melovské frekvenční kepstrální koeficienty (z angličtiny – Mel Frequency Cepstrum Coefficients)
PLP	Perceptivní lineární prediktivní (analýza) [z angličtiny – Perceptual Linear Predictive (analysis)]
TTS	Syntéza řeči z textu (z angličtiny – Text-To-Speech)
FFT	Rychlá Fourierova transformace (z angličtiny – Fast Fourier Transform)
DTW	Dynamické borcení času (z angličtiny – Dynamic Time Warping)
FST	Konečný transducer (z angličtiny – Finite State Transducer)
WFST	Vážený konečný transducer (z angličtiny – Weighted Finite State Transducer)
CMVN	Normalizace kepstrální střední hodnoty a variance (z angličtiny – Cepstral Mean and Variance Normalization)
MLF	Master Label File

Literatura

[1] BIGI, B. SPPAS: a tool for the phonetic segmentations of speech. In *Proceedings of LREC*, Istanbul, Turecko, 2012, s. 1748–1755.

[2] BOERSMA, P., VAN HEUVEN, V. Speak and unSpeak with PRAAT. *Glot International*. 2001, **5**(9), s. 341–347.

[3] COSI, P., GALATÀ, V., CUTUGNO, F., aj. Forced Alignment on Children Speech. In *Proceedings of CLiC-it and EVALITA*, Pisa, Itálie, 2014, s. 124–126.

[4] ŽELEZNÝ, M. *Strukturální metody rozpoznávání. (skripta)*. ZČU Plzeň.

[5] GOLDMAN, J.-P. EasyAlign: an automatic phonetic alignment tool under Praat. In *Proceedings of INTERSPEECH*, Florencie, Itálie, 2011.

[6] JURAFSKY, D., MARTIN, H. *Speech and Language Processing*. New Jersey: Pearson Education, 2009, ISBN 978-0-13-187321-6.

[7] KIM, Y.-J., CONKIE, A. Automatic segmentation combining an HMM-based approach and spectral boundary correction. In *Proceedings of INTERSPEECH*, Denver, USA, 2002, s. 145–148.

[8] KOMINEK, J., BENNET, C. L., BLACK, A. W. Evaluating and correcting phoneme segmentation for unit selection synthesis. In *Proceedings of INTERSPEECH*, Ženeva, Švýcarsko, 2003, s. 313–316.

[9] KORVAS, M., PLÁTEK, O., DUŠEK, O., aj. Free English and Czech telephone speech corpus shared under the CC-BY-SA 3.0 license. In *Proceedings of LREC*, Reykjavik, Island, 2014, s. 4423–4428.

[10] LEE, A., KAWAHARA, T., SHIKANO, K. Julius–an open source real-time large vocabulary recognition engine. In *Proceedings of EUROSPEECH*, Aalborg, Dánsko, 2001, s. 1691–1694.

[11] MATOUŠEK, J. *Počítačová syntéza řeči*. Plzeň. 2008. Habilitační práce. Západočeská univerzita v Plzni.

[12] MATOUŠEK, J., TIHELKA, D., PSUTKA, J. Automatic segmentation for Czech concatenative speech synthesis using statistical approach with boundary-specific correction. In *Proceedings of INTERSPEECH*, Ženeva, Švýcarsko, 2003, s. 301–304.

[13] MATOUŠEK, J., TIHELKA, D., PSUTKA, J. Experiments with automatic segmentation for Czech speech synthesis. *Lecture Notes in Computer Science*, Berlín:Springer, 2003, **2607**, s. 287–294.

[14] MATOUŠEK, J., TIHELKA, D., ROMPORTL, J. Building of a speech corpus optimised for unit selection TTS synthesis. In *Proceedings of LREC*, Marrákéš, Maroko, 2008.

[15] MOHRI, M., PEREIRA, F., RILEY, M. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, Amsterdam:Elsevier, 2002, **16**(1), 69–88.

[16] OGBUREKE, K., Carson–Berndsen, J. Improving initial boundary estimation for HMM-based automatic phonetic segmentation. In *Proceedings of INTERSPEECH*, Brighton, UK, 2009, s. 884–887.

[17] PATČ, Z., MIZERA, P., POLLAK, P. Phonetic Segmentation Using KALDI and Reduced Pronunciation Detection in Casual Czech Speech. In *Proceedings of Text, Speech, and Dialogue*, Plzeň,

[18] POVEY, D., GHOSHAL, A., BOULIANNE, G., aj. The Kaldi speech recognition toolkit. In *Proceedings of IEEE ASRU*, Waikoloa, USA, 2011.

[19] PSUTKA, J., MÜLLER, L., MATOUŠEK, J., RADOVÁ, V. *Mluvíme s počítačem česky*. Praha: Academia, 2006, ISBN 8020013091.

[20] RENDEL, A., SORIN, A., HOORY, R., aj. Towards automatic phonetic segmentation for TTS. In *Proceedings of ICASSP*, Kyoto, Japonsko, 2012, s. 4533–4536.

[21] ROSILLO, G. V. *Automatic speech recognition with Kaldi toolkit*. Barcelona. 2016. Degree Final Project. Universitat Politècnica de Catalunya.

[22] STRAND, O., EGEBERG, A. Cepstral mean and variance normalization in the model domain. In *Proceedings of ISCA ITRW*, Norwich, UK, 2004.

[23] TAYLOR, P. *Text-to-Speech Synthesis*. New York: Cambridge University Press, 2009, ISBN 978-0-521-89927-7.

[24] YAMAGISHI, J. *An introduction to hmm-based speech synthesis*. Tokio. 2006. Technická zpráva. Tokyo Institute of Technology.

[25] YOUNG, S., KERSHAW, D., ODELL, J., aj. *The HTK Book (for HTK Version 3.4)* [online]. Cambridge University Engineering Department, 2006 [cit. 2.5.2016]. Dostupné z http://speech.ee.ntu.edu.tw/homework/DSP_HW2-1/htkbook.pdf.